

# 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2020, December 14–18, 2020, BITS Pilani,  
K K Birla Goa Campus, Goa, India (Virtual Conference)

Edited by

Nitin Saxena

Sunil Simon



*Editors*

**Nitin Saxena** 

Indian Institute of Technology Kanpur, India  
nitin@cse.iitk.ac.in

**Sunil Simon** 

Indian Institute of Technology Kanpur, India  
simon@cse.iitk.ac.in

*ACM Classification 2012*

Theory of computation; Computing methodologies; Software and its engineering

**ISBN 978-3-95977-174-0**

*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-174-0>.

*Publication date*

December, 2020

*Bibliographic information published by the Deutsche Nationalbibliothek*

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

*License*

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):  
<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2020.0

ISBN 978-3-95977-174-0

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

## LIPICs – Leibniz International Proceedings in Informatics

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

### *Editorial Board*

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

**ISSN 1868-8969**

**<https://www.dagstuhl.de/lipics>**



## ■ Contents

Preface	
<i>Nitin Saxena and Sunil Simon</i> .....	0:ix

### Invited Talks

The Quest for Mathematical Understanding of Deep Learning	
<i>Sanjeev Arora</i> .....	1:1–1:1
Proofs of Soundness and Proof Search	
<i>Albert Atserias</i> .....	2:1–2:1
Convex Optimization and Dynamic Data Structure	
<i>Yin Tat Lee</i> .....	3:1–3:1
Holonomic Techniques, Periods, and Decision Problems	
<i>Joël Ouaknine</i> .....	4:1–4:3
Algorithmic Improvisation for Dependable Intelligent Autonomy	
<i>Sanjit A. Seshia</i> .....	5:1–5:3
On Some Recent Advances in Algebraic Complexity	
<i>Amir Shpilka</i> .....	6:1–6:1

### Regular Papers

Faster Property Testers in a Variation of the Bounded Degree Model	
<i>Isolde Adler and Polly Fahey</i> .....	7:1–7:15
Clustering Under Perturbation Stability in Near-Linear Time	
<i>Pankaj K. Agarwal, Hsien-Chih Chang, Kamesh Munagala, Erin Taylor, and Emo Welzl</i> .....	8:1–8:16
Width Notions for Ordering-Related Problems	
<i>Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, and Petra Wolf</i> .....	9:1–9:18
Optimal Output Sensitive Fault Tolerant Cuts	
<i>Niranka Banerjee, Venkatesh Raman, and Saket Saurabh</i> .....	10:1–10:19
Online Matching with Recourse: Random Edge Arrivals	
<i>Aaron Bernstein and Aditi Dudeja</i> .....	11:1–11:16
Hard QBFs for Merge Resolution	
<i>Olaf Beyersdorff, Joshua Blinkhorn, Meena Mahajan, Tomáš Peitl, and Gaurav Sood</i> .....	12:1–12:15
On Sampling Based Algorithms for $k$ -Means	
<i>Anup Bhattacharya, Dishant Goyal, Ragesh Jaiswal, and Amit Kumar</i> .....	13:1–13:17
String Indexing for Top- $k$ Close Consecutive Occurrences	
<i>Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, Eva Rotenberg, and Teresa Anna Steiner</i> .....	14:1–14:17

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).  
Editors: Nitin Saxena and Sunil Simon



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

Fair Tree Connection Games with Topology-Dependent Edge Cost <i>Davide Bilò, Tobias Friedrich, Pascal Lenzner, Anna Melnichenko, and Louise Molitor</i> .....	15:1–15:15
Locally Decodable/Correctable Codes for Insertions and Deletions <i>Alexander R. Block, Jeremiah Blocki, Elena Grigorescu, Shubhang Kulkarni, and Minshen Zhu</i> .....	16:1–16:17
Maximum Clique in Disk-Like Intersection Graphs <i>Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow</i> .....	17:1–17:18
Parameterized Complexity of Feedback Vertex Sets on Hypergraphs <i>Pratibha Choudhary, Lawqueen Kanesh, Daniel Lokshantov, Fahad Panolan, and Saket Saurabh</i> .....	18:1–18:15
Size Bounds on Low Depth Circuits for Promise Majority <i>Joshua Cook</i> .....	19:1–19:14
Lower Bounds for Semi-adaptive Data Structures via Corruption <i>Pavel Dvořák and Bruno Loff</i> .....	20:1–20:15
Stability-Preserving, Time-Efficient Mechanisms for School Choice in Two Rounds <i>Karthik Gajulapalli, James A. Liu, Tung Mai, and Vijay V. Vazirani</i> .....	21:1–21:15
New Verification Schemes for Frequency-Based Functions on Data Streams <i>Prantar Ghosh</i> .....	22:1–22:15
Online Carpooling Using Expander Decompositions <i>Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Sahil Singla</i> .....	23:1–23:14
On the (Parameterized) Complexity of Almost Stable Marriage <i>Sushmita Gupta, Pallavi Jain, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi</i> ..	24:1–24:17
Min-Cost Popular Matchings <i>Telikepalli Kavitha</i> .....	25:1–25:17
Constructing Large Matchings via Query Access to a Maximal Matching Oracle <i>Lidiya Khalidah binti Khalil and Christian Konrad</i> .....	26:1–26:15
Planted Models for the Densest $k$ -Subgraph Problem <i>Yash Khanna and Anand Louis</i> .....	27:1–27:18
Sample-And-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model <i>Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju</i> .....	28:1–28:18
On Parity Decision Trees for Fourier-Sparse Boolean Functions <i>Nikhil S. Mande and Swagato Sanyal</i> .....	29:1–29:16
Colored Cut Games <i>Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer</i> ..	30:1–30:17
Randomness Efficient Noise Stability and Generalized Small Bias Sets <i>Dana Moshkovitz, Justin Oh, and David Zuckerman</i> .....	31:1–31:16
Connectivity Lower Bounds in Broadcast Congested Clique <i>Shreyas Pai and Sriram V. Pemmaraju</i> .....	32:1–32:17

Fully Dynamic Sequential and Distributed Algorithms for MAX-CUT <i>Omer Wasim and Valerie King</i> .....	33:1–33:19
Weighted Tiling Systems for Graphs: Evaluation Complexity <i>C. Aiswarya and Paul Gastin</i> .....	34:1–34:17
Process Symmetry in Probabilistic Transducers <i>Shaul Almagor</i> .....	35:1–35:14
Reachability in Dynamical Systems with Rounding <i>Christel Baier, Florian Funke, Simon Jantsch, Toghrol Karimov, Engel Lefaucheur, Joël Ouaknine, Amaury Pouly, David Purser, and Markus A. Whiteland</i> .....	36:1–36:17
Parameterized Complexity of Safety of Threshold Automata <i>A. R. Balasubramanian</i> .....	37:1–37:15
Uncertainty Reasoning for Probabilistic Petri Nets via Bayesian Networks <i>Rebecca Bernemann, Benjamin Cabrera, Reiko Heckel, and Barbara König</i> .....	38:1–38:17
Synthesizing Safe Coalition Strategies <i>Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar</i> .....	39:1–39:17
Dynamic Network Congestion Games <i>Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur</i> .....	40:1–40:16
On the Succinctness of Alternating Parity Good-For-Games Automata <i>Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak</i> .....	41:1–41:13
A Framework for Consistency Algorithms <i>Peter Chini and Prakash Saivasan</i> .....	42:1–42:17
Equivalence of Hidden Markov Models with Continuous Observations <i>Oscar Darwin and Stefan Kiefer</i> .....	43:1–43:14
Nivat-Theorem and Logic for Weighted Pushdown Automata on Infinite Words <i>Manfred Droste, Sven Dziadek, and Werner Kuich</i> .....	44:1–44:14
Synchronization of Deterministic Visibly Push-Down Automata <i>Henning Fernau and Petra Wolf</i> .....	45:1–45:15
Synthesis from Weighted Specifications with Partial Domains over Finite Words <i>Emmanuel Filiot, Christof Löding, and Sarah Winter</i> .....	46:1–46:16
Reachability for Updatable Timed Automata Made Faster and More Effective <i>Paul Gastin, Sayan Mukherjee, and B. Srivathsan</i> .....	47:1–47:17
Active Prediction for Discrete Event Systems <i>Stefan Haar, Serge Haddad, Stefan Schwoon, and Lina Ye</i> .....	48:1–48:16
Comparing Labelled Markov Decision Processes <i>Stefan Kiefer and Qiyi Tang</i> .....	49:1–49:16
Computable Analysis for Verified Exact Real Computation <i>Michal Konečný, Florian Steinberg, and Holger Thies</i> .....	50:1–50:18
Perspective Games with Notifications <i>Orna Kupferman and Noam Shenwald</i> .....	51:1–51:16

On the Complexity of Multi-Pushdown Games <i>Roland Meyer and Sören van der Wall</i> .....	52:1–52:35
Higher-Order Nonemptiness Step by Step <i>Paweł Parys</i> .....	53:1–53:14
The Degree of a Finite Set of Words <i>Dominique Perrin and Andrew Ryzhikov</i> .....	54:1–54:16
What You Must Remember When Transforming Datawords <i>M. Praveen</i> .....	55:1–55:14
Minimising Good-For-Games Automata Is NP-Complete <i>Sven Schewe</i> .....	56:1–56:13
Static Race Detection for RTOS Applications <i>Rishi Tulsyan, Rekha Pai, and Deepak D’Souza</i> .....	57:1–57:20
Synchronization Under Dynamic Constraints <i>Petra Wolf</i> .....	58:1–58:14

## ■ Preface

This volume contains the proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020). The conference was originally planned to be held in BITS Pilani, K K Birla Goa Campus, Goa, India. Due to the COVID-19 pandemic, the conference was held online from December 15 to December 17, 2020. The conference had two tracks, Track-A focussing on algorithms, complexity and related issues and Track-B focussing on logic, automata and other formal methods aspects of computer science. Each track had its own Program Committee (PC) with a chair. This volume constitutes the joint proceedings of the two tracks, published in the LIPIcs series under a Creative Commons license, with free online access to all.

The conference comprised of 6 invited talks, 27 contributed talks in Track-A and 25 in Track-B. This volume contains all the contributed papers from both tracks and the abstracts of all invited talks presented at the conference. There were overall 138 submissions, 75 in Track-A and 63 in Track-B. We thank all the authors who submitted their papers to FSTTCS 2020. We also express our gratitude to all the PC members for their tireless work and all external reviewers for their expert opinion in the form of timely reviews.

We are grateful to all the invited speakers: Sanjeev Arora (Princeton University, U.S.A.), Albert Atserias (Universitat Politècnica de Catalunya, Spain), Yin-Tat Lee (University of Washington, U.S.A.), Joël Ouaknine (MPI for Software Systems, Saarbrücken, Germany and University of Oxford, U.K.), Sanjit Seshia (University of California, Berkeley, U.S.A.) and Amir Shpilka (Tel Aviv University, Israel). They kindly accepted our invitations and gave talks that inspired the entire audience.

The main conference was preceded by two workshops: *Workshop on Matrix Rigidity*, organised by Amey Bhangale (University of California, Riverside), Alexander Golovnev (Georgetown University), Mrinal Kumar (IIT Bombay) and Amit Kumar Sinhababu (University of Ulm, Germany), and *Strategies for Uncertainty (SUN)* organised by Dietmar Berwanger (CNRS & ENS Paris-Saclay) and R. Ramanujam (IMSc, Chennai). This was followed by a post-conference workshop: *Advances in Verification* organised by Prakash Saivasan (IMSc, Chennai) and B. Srivathsan (CMI, Chennai). In addition, there was a co-located event: *Workshop on Research Highlights in Programming Languages* organised by Deepak D'Souza (IISc, Bangalore), Uday P. Khedkar (IIT Bombay), K. Narayan Kumar (CMI, Chennai), Komondoor V. Raghavan (IISc, Bangalore) and Aseem Rastogi (Microsoft Research, India).

We are indebted to the organising committee members: A. Baskar (BITS Pilani), Pritam Bhattacharya (BITS Pilani), Amaldev Manuel (IIT Goa), Anup Basil Mathew (BITS Pilani) and A.V. Sreejith (IIT Goa). They ensured a smooth running of the conference and workshops; and made all necessary arrangements to shift to the online mode. We thank S.P. Suresh (CMI, Chennai) for maintaining the conference webpage and promptly addressing our update requests. We also thank the friendly staff at Dagstuhl LIPIcs, particularly Michael Wagner, for being prompt and helpful in answering our queries. Finally, we thank the members of the Steering Committee for having confidence in us for running the conference; and giving us pertinent advice to handle the unprecedented changes made in the conduct of the conference.

Nitin Saxena and Sunil Simon  
December 2020





## ■ Program Committee

### Track A

- Siddharth Barman (Indian Institute of Science, Bangalore)
- Markus Blaeser (Saarland University)
- Chandra Chekuri (University of Illinois, Urbana-Champaign)
- Zeev Dvir (Princeton University)
- Ankit Garg (Microsoft Research India)
- Rohit Gurjar (Indian Institute of Technology Bombay)
- Rahul Jain (National University of Singapore)
- Ravindran Kannan (Microsoft Research India)
- Soumen Maity (IISER Pune)
- Denis Pankratov (Concordia University)
- Rahul Santhanam (University of Oxford)
- Nitin Saxena (Indian Institute of Technology Kanpur) – **co-chair**
- C Seshadhri (University of California, Santa Cruz)
- Jiri Sgall (Charles University, Prague)
- Raghunath Tewari (Indian Institute of Technology Kanpur)
- Ramarathnam Venkatesan (Microsoft Research India)
- Ronald de Wolf (CWI & University of Amsterdam, The Netherlands)

### Track B

- Dietmar Berwanger (LSV, CNRS and ENS Paris-Saclay)
- Mikołaj Bojanczyk (University of Warsaw)
- Benedikt Bollig (LSV, CNRS and ENS Paris-Saclay)
- Rohit Chadha (University of Missouri)
- Supratik Chakraborty (Indian Institute of Technology Bombay)
- Ranko Lazic (University of Warwick)
- Amaldev Manuel (Indian Institute of Technology Goa)
- Roland Meyer (TU Braunschweig)
- Angelo Montanari (University of Udine)
- Andrzej Murawski (University of Oxford)
- Anca Muscholl (LaBRI, Université Bordeaux)
- Daniel Neider (MPI for Software Systems, Kaiserslautern)
- Komondoor V Raghavan (Indian Institute of Science Bangalore)
- Aseem Rastogi (Microsoft Research India)
- Sunil Simon (Indian Institute of Technology Kanpur) – **co-chair**
- S P Suresh (Chennai Mathematical Institute)
- Aditya Thakur (University of California, Davis)





## ■ List of Reviewers: Track-A

A. Karim Abu-Affash  
Akanksha Agrawal  
Amit Deshpande  
Amit Kumar  
Andreas Emil Feldmann  
Anil Shukla  
Anuj Dawar  
Aritra Banik  
Arnab Bhattacharya (IITK)  
Arnau Messegué  
Benjamin Raichel  
Bhargav Thankey  
Christian Sohler  
Deeparnab Chakrabarty  
Dominik Kempa  
Gregory Wilsenach  
Hendrik Fichtenberger  
Ildikó Schlotter  
Janardhan Kulkarni  
Jayalal Sarma  
Jonathan Lee  
Justin Thaler  
Karthikeyan Chandrasekaran  
Krzysztof Nowicki  
Mariusz Jakubowski  
Maximilian Katzmann  
Mikolas Janota  
Morteza Monemizadeh  
Naoyuki Kamiyama  
Navin Goyal  
Nidhi Rathi  
Nishanth Chandran  
Palash Dey  
Patrick Rall  
Peng Zhang  
Pranjal Awasthi  
Rahul Madhavan  
Ramanujan M Sridharan  
Sagar Kale  
Saladi Rahul  
Satyanarayana Lokam  
Shachar Lovett  
Shalev Ben-David  
Shivika Narang  
Srijita Kundu  
Ahmad Biniiaz  
Alireza Farhadi  
Amit Kumar  
Anat Paskin-Cherniavsky  
Andrew McGregor  
Anna Gal  
Aravindan Vijayaraghavan  
Arkadev Chattopadhyay  
Arnab Bhattacharyya (NUS)  
Ben Lee Volk  
Bhargav Narayanan  
Christian Konrad  
Clément Dallard  
Diptarka Chakraborty  
Geevarghese Philip  
Hamed Saleh  
Hiro Ito  
Ivor van der Hoog  
Jason Li  
Jesper Nederlof  
Julian Dörfler  
Karl Bringmann  
Kartik Gupta  
Krzysztof Onak  
Martin Koutecky  
Mehtaab Sawhney  
Mohit Singh  
Mrinal Kumar  
Naresh Goud Boddu  
Neeraj Kayal  
Ning Xie  
Omri Weinstein  
Pat Morin  
Pavel Hubáček  
Prajakta Nimbhorkar  
Ragesh Jaiswal  
Rajat Mittal  
Roshan Raj  
Saket Saurabh  
Sanjeev Khanna  
Sean Kafer  
Shahbaz Khan  
Sharma V Thankachan  
Spyridon Tzimas  
Srikanth Srinivasan

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).  
Editors: Nitin Saxena and Sunil Simon



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

Srinivasan Raghuraman  
Stefan Mengel  
Subrahmanyam Kalyanasundaram  
Suman Kalyan Bera  
Swagato Sanyal  
Tasuku Soma  
Thatchaphol Saranurak  
Tobias Mömke  
V Vinay  
Yash Khanna  
Yuichi Yoshida

Sriram Rajamani  
Stefano Leucci  
Sudeshna Kolay  
Suprovat Ghoshal  
Tara Abrishami  
Telikepalli Kavitha  
Theo McKenzie  
Upendra Kapshikar  
Vishakha Patil  
Yassine Hamoudi  
Yuval Filmus

## ■ List of Reviewers: Track-B

C. Aiswarya  
Shaull Almagor  
Nikhil Balaji  
Massimo Benerecetti  
Davide Bresolin  
Peter Chini  
Alberto Ciaffaglione  
Deepak D'Souza  
Bart de Keijzer  
Giorgio Delzanno  
David Doty  
Goran Frehse  
Luca Geatti  
Shibashis Guha  
Thomas Haas  
Loic Helouet  
Stefan Kiefer  
Denis Kuperberg  
Christof Löding  
Kaushik Mallik  
Filip Mazowiecki  
Sebastian Muskalla  
Guillermo Perez  
Carla Piazza  
Pavithra Prabhakar  
David Purser  
R. Ramanujam  
Nima Roohi  
Krishna S  
Pietro Sala  
Sriram Sankaranarayanan  
Natarajan Shankar  
Kevin Stier  
Vaishnavi Sundararajan  
Ramanathan Thinniyam Srinivasan  
Ashutosh Trivedi  
Przemysław Andrzej Wałęga  
Cas Widdershoven  
Sebastian Wolff  
Martin Zimmermann

S. Akshay  
Kazuyuki Asada  
Hugo Bazille  
Udi Boker  
Sourav Chakraborty  
Dmitry Chistikov  
Wojciech Czerwiński  
Laure Daviaud  
Dario Della Monica  
Alex Dixon  
Nathanaël Fijalkow  
Matthias Függer  
Nicola Gigante  
Jens Oliver Gutsfeld  
Peter Habermehl  
Ismaël Jecker  
Igor Konnov  
Salvatore La Torre  
Meena Mahajan  
Konstantinos Mamouras  
Benjamin Monmege  
Youssef Oualhadj  
Dominique Perrin  
Anton Pirogov  
M. Praveen  
Karin Quaas  
Cristian Riveros  
Subhajit Roy  
Prakash Saivasan  
Arnaud Sangnier  
Sven Schewe  
B Srivathsan  
Howard Straubing  
Nathalie Sznajder  
Patrick Totzke  
Sören van der Wall  
Pascal Weil  
Sarah Winter  
Xiang Yin





# The Quest for Mathematical Understanding of Deep Learning

Sanjeev Arora

Computer Science Department, Princeton University, NJ, USA

<https://www.cs.princeton.edu/~arora/>

arora@cs.princeton.edu

---

## Abstract

Deep learning has transformed Machine Learning and Artificial Intelligence in the past decade. It raises fundamental questions for mathematics and theory of computer science, since it relies upon solving large-scale nonconvex problems via gradient descent and its variants. This talk will be an introduction to mathematical questions raised by deep learning, and some partial understanding obtained in recent years.

**2012 ACM Subject Classification** Theory of computation → Mathematical optimization; Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning

**Keywords and phrases** machine learning, artificial intelligence, deep learning, gradient descent, optimization

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.1

**Category** Invited Talk



© Sanjeev Arora;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 1; pp. 1:1–1:1



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Proofs of Soundness and Proof Search

Albert Atserias 

Universitat Politècnica de Catalunya, Barcelona, Spain  
atserias@cs.upc.edu

---

## Abstract

Let  $P$  be a sound proof system for propositional logic. For each CNF formula  $F$ , let  $\text{SAT}(F)$  be a CNF formula whose satisfying assignments are in 1-to-1 correspondence with those of  $F$  (e.g.,  $F$  itself). For each integer  $s$ , let  $\text{REF}(F, s)$  be a CNF formula whose satisfying assignments are in 1-to-1 correspondence with the  $P$ -refutations of  $F$  of length  $s$ . Since  $P$  is sound, it is obvious that the conjunction formula  $\text{SAT}(F) \ \& \ \text{REF}(F, s)$  is unsatisfiable for any choice of  $F$  and  $s$ . It has been long known that, for many natural proof systems  $P$  and for the most natural formalizations of the formulas  $\text{SAT}$  and  $\text{REF}$ , the unsatisfiability of  $\text{SAT}(F) \ \& \ \text{REF}(F, s)$  can be established by a short refutation. In addition, for many  $P$ , these short refutations live in the proof system  $P$  itself. This is the case for all Frege proof systems. In contrast it was known since the early 2000's that Resolution proofs of Resolution's soundness statements must have superpolynomial length. In this talk I will explain how the soundness formulas for a proof system  $P$  relate to the computational complexity of the proof search problem for  $P$ . In particular, I will explain how such formulas are used in the recent proof that the problem of approximating the minimum proof-length for Resolution is NP-hard (Atserias-Müller 2019). Besides playing a key role in this hardness of approximation result, the renewed interest in the soundness formulas led to a complete answer to the question whether Resolution has subexponential-length proofs of its own soundness statements (Garlík 2019).

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Automated reasoning

**Keywords and phrases** Proof complexity, automatability, Resolution, proof search, consistency statements, lower bounds, reflection principle, satisfiability

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.2

**Category** Invited Talk



© Albert Atserias;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 2; pp. 2:1–2:1



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Convex Optimization and Dynamic Data Structure

**Yin Tat Lee**

University of Washington, Seattle, WA, USA

<http://yintat.com>

[yintat@uw.edu](mailto:yintat@uw.edu)

---

## Abstract

In the last three years, there are many breakthroughs in optimization such as nearly quadratic time algorithms for bipartite matching, linear programming algorithms that are as fast as  $Ax = b$ . All of these algorithms are based on a careful combination of optimization techniques and dynamic data structures. In this talk, we will explain the framework underlying all the recent breakthroughs.

Joint work with Jan van den Brand, Michael B. Cohen, Sally Dong, Haotian Jiang, Tarun Kathuria, Danupon Nanongkai, Swati Padmanabhan, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, Di Wang, Sam Chiu-wai Wong, Guanghao Ye, Qiuyi Zhang.

**2012 ACM Subject Classification** Mathematics of computing → Mathematical optimization

**Keywords and phrases** Convex Optimization, Dynamic Data Structure

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.3

**Category** Invited Talk



© Yin Tat Lee;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 3; pp. 3:1–3:1



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Holonomic Techniques, Periods, and Decision Problems

Joël Ouaknine 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany  
Department of Computer Science, Oxford University, UK

---

## Abstract

Holonomic techniques have deep roots going back to Wallis, Euler, and Gauss, and have evolved in modern times as an important subfield of computer algebra, thanks in large part to the work of Zeilberger and others over the past three decades. In this talk, I will give an overview of the area, and in particular will present a select survey of known and original results on decision problems for holonomic sequences and functions. (*Holonomic sequences* satisfy linear recurrence relations with polynomial coefficients, and *holonomic functions* satisfy linear differential equations with polynomial coefficients.) I will also discuss some surprising connections to the theory of periods and exponential periods, which are classical objects of study in algebraic geometry and number theory; in particular, I will relate the decidability of certain decision problems for holonomic sequences to deep conjectures about periods and exponential periods, notably those due to Kontsevich and Zagier.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** holonomic techniques, decision problems, recurrence sequences, minimal solutions, Positivity Problem, continued fractions, special functions, periods, exponential periods

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.4

**Category** Invited Talk

**Funding** *Joël Ouaknine*: Supported by ERC grant AVS-ISS (648701) and by DFG grant 389792660 as part of TRR 248 (see <https://perspicuous-computing.science>).

## 1 Summary

*Holonomic sequences* (also known as *P-recursive* or *P-finite* sequences) are infinite sequences of real (or complex) numbers that satisfy a linear recurrence relation with polynomial coefficients. The earliest and best-known example is the Fibonacci sequence, introduced by Leonardo of Pisa in the 12th century; more recently, Apéry famously made use of certain holonomic sequences  $\langle u_n \rangle_n$  satisfying the recurrence relation

$$(n+1)^3 u_{n+1} = (34n^3 + 51n^2 + 27n + 5)u_n - n^3 u_{n-1} \quad (n \in \mathbb{N})$$

to prove that  $\zeta(3) := \sum_{n=1}^{\infty} n^{-3}$  is irrational [2]. Holonomic sequences now form a vast subject in their own right, with numerous applications in mathematics and other sciences; see, for instance, the monographs [20, 5, 6] or the seminal paper [24] of Zeilberger.

Any holonomic sequence  $\langle u_n \rangle_{n=0}^{\infty}$  naturally gives rise to a *holonomic function* by considering the associated generating power series  $\mathcal{F}(x) = \sum_{n=0}^{\infty} u_n x^n$ . The recurrence relation defining the holonomic sequence in turn yields a linear differential equation satisfied by the corresponding power series.

There is a voluminous literature devoted to the study of identities for holonomic sequences and functions, and several computer-algebra packages implementing various identity-checking algorithms are also available. However, as noted by Kauers and Pillwein, “*in contrast, [...] almost no algorithms are available for inequalities*” [11]. For example, the *Positivity Problem*



© Joël Ouaknine;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 4; pp. 4:1–4:3



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(i.e., whether every term of a given sequence is non-negative) for  $C$ -finite sequences<sup>1</sup> is only known to be decidable at low orders, and there is strong evidence that the problem is mathematically intractable in general [19, 18]; see also [10, 14, 19, 17]. For holonomic sequences that are not  $C$ -finite, virtually no decision procedures currently exist for Positivity, although several partial results and heuristics are known (see, for example [15, 11, 16, 23, 21, 22]).

Another extremely important property of holonomic sequences is *minimality*; a sequence  $\langle u_n \rangle_n$  is minimal if, given any other linearly independent sequence  $\langle v_n \rangle_n$  satisfying the same recurrence relation, the ratio  $u_n/v_n$  converges to 0. Minimal holonomic sequences play a crucial rôle, among others, in numerical calculations and asymptotics, as noted for example in [7, 8, 9, 3, 1, 4] – see also the references therein. Unfortunately, there is also ample evidence that determining algorithmically whether a given holonomic sequence is minimal is a very challenging task, for which no satisfactory solution is at present known to exist.

In this talk, I will present a select survey of known and original results on decision problems for holonomic sequences and functions. Some of this work will involve *periods* and *exponential periods*, which are classical objects of study in algebraic geometry and number theory; in particular, I will relate the decidability of certain decision problems for holonomic sequences to deep conjectures about periods and exponential periods, notably those due to Kontsevich and Zagier [13]. Parts of this presentation will be based on the paper [12].

---

## References

- 1 Gil Amparo, Javier Segura, and Nico M. Temme. Numerical methods for special functions, 2007.
- 2 Roger Apéry. Irrationalité de  $\zeta(2)$  et  $\zeta(3)$ . In *Journées Arithmétiques de Luminy*, number 61 in Astérisque, pages 11–13. Société mathématique de France, 1979. URL: [http://www.numdam.org/item/AST\\_1979\\_\\_61\\_\\_11\\_0](http://www.numdam.org/item/AST_1979__61__11_0).
- 3 Alfredo Deaño and Javier Segura. Transitory minimal solutions of hypergeometric recursions and pseudoconvergence of associated continued fractions. *Mathematics of Computation*, 76(258):879–901, 2007.
- 4 Alfredo Deaño, Javier Segura, and Nico M. Temme. Computational properties of three-term recurrence relations for Kummer functions. *J. Computational Applied Mathematics*, 233(6):1505–1510, 2010.
- 5 Graham Everest, Alfred J. van der Poorten, Igor E. Shparlinski, and Thomas Ward. *Recurrence Sequences*, volume 104 of *Mathematical surveys and monographs*. American Mathematical Society, 2003.
- 6 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
- 7 Walter Gautschi. Computational aspects of three-term recurrence relations. *SIAM Rev.*, 9:24–82, 1967.
- 8 Walter Gautschi. Anomalous convergence of a continued fraction for ratios of kummer functions. *Mathematics of Computation*, 31(140):994–999, 1977.
- 9 Walter Gautschi. Minimal solutions of three-term recurrence relations and orthogonal polynomials. *Mathematics of Computation*, 36(154), 1981.
- 10 V. Halava, T. Harju, and M. Hirvensalo. Positivity of second order linear recurrent sequences. *Discrete Appl. Math.*, 154(3):447–451, 2006.

---

<sup>1</sup>  $C$ -finite sequences are linear recurrent sequences with *constant* coefficients.

- 11 Manuel Kauers and Veronika Pillwein. When can we detect that a P-finite sequence is positive? In Wolfram Koepf, editor, *Symbolic and Algebraic Computation, International Symposium, ISSAC 2010, Munich, Germany, July 25-28, 2010, Proceedings*, pages 195–201. ACM, 2010.
- 12 George Kenison, Oleksiy Klurman, Engel Lefauchaux, Florian Luca, Pieter Moree, Joël Ouaknine, Markus A. Whiteland, and James Worrell. On positivity and minimality for second-order holonomic sequences. *CoRR*, abs/2007.12282, 2020. URL: <https://arxiv.org/abs/2007.12282>.
- 13 Maxim Kontsevich and Don Zagier. Periods. In *Mathematics unlimited—2001 and beyond*, pages 771–808. Springer, Berlin, 2001.
- 14 V. Laohakosol and P. Tangsupphathawat. Positivity of third order linear recurrence sequences. *Discrete Appl. Math.*, 157(15):3239–3248, 2009.
- 15 Lily Liu. Positivity of three-term recurrence sequences. *Electron. J. Combin.*, 17(1):Research Paper 57, 10, 2010.
- 16 M. Mezzarobba and B. Salvy. Effective bounds for P-recursive sequences. *J. Symbolic Comput.*, 45(10):1075–1096, 2010.
- 17 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2014.
- 18 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015.
- 19 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 366–379. ACM, New York, 2014.
- 20 Marko Petkovšek, Herbert Wilf, and Doron Zeilberger. *A=B*. A. K. Peters, 1997.
- 21 Veronika Pillwein. Termination conditions for positivity proving procedures. In Manuel Kauers, editor, *International Symposium on Symbolic and Algebraic Computation, ISSAC'13, Boston, MA, USA, June 26-29, 2013*, pages 315–322. ACM, 2013.
- 22 Veronika Pillwein and Miriam Schussler. An efficient procedure deciding positivity for a class of holonomic functions. *ACM Comm. Computer Algebra*, 49(3):90–93, 2015.
- 23 Ernest X. W. Xia and X. M. Yao. The signs of three-term recurrence sequences. *Discrete Applied Mathematics*, 159(18):2290–2296, 2011.
- 24 Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.



# Algorithmic Improvisation for Dependable Intelligent Autonomy

Sanjit A. Seshia 

University of California, Berkeley, CA, USA

<http://www.eecs.berkeley.edu/~sseshia>

sseshia@eecs.berkeley.edu

---

## Abstract

Algorithmic Improvisation, also called control improvisation or controlled improvisation, is a new framework for automatically synthesizing systems with specified random but controllable behavior. In this talk, I will present the theory of algorithmic improvisation and show how it can be used in a wide variety of applications where randomness can provide variety, robustness, or unpredictability while guaranteeing safety or other properties. Applications demonstrated to date include robotic surveillance, software fuzz testing, music improvisation, human modeling, generating test cases for simulating cyber-physical systems, and generation of synthetic data sets to train and test machine learning algorithms. In this talk, I will particularly focus on applications to the design of intelligent autonomous systems, presenting work on randomized planning for robotics and a domain-specific probabilistic programming language for the design and analysis of learning-based autonomous systems.

**2012 ACM Subject Classification** Theory of computation → Automated reasoning; Computing methodologies → Machine learning; Computing methodologies → Artificial intelligence; Theory of computation → Formal languages and automata theory; Software and its engineering → Domain specific languages; Theory of computation → Complexity theory and logic

**Keywords and phrases** Formal methods, synthesis, verification, randomized algorithms, formal specification, testing, machine learning, synthetic data generation, planning

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.5

**Category** Invited Talk

**Funding** This work has been supported in part by the National Science Foundation (NSF) Graduate Research Fellowship Program under Grant No. DGE1106400, by NSF grants CCF-1139138, CNS-1545126 (VeHiCaL), CNS-1646208, and CCF-1837132, DARPA under agreement numbers FA8750-16-C0043 and FA8750-18-C-0101, by the iCyPhy center, Berkeley Deep Drive, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

**Acknowledgements** The work on algorithmic improvisation described in this talk has been co-led by my Ph.D. student, Daniel Fremont, and is reported in his Ph.D. dissertation.

## Summary

*Algorithmic improvisation* is a new framework for automatically synthesizing systems with specified random but controllable behavior. Such systems are known as improvisers and have applications in a variety of applications where randomness can provide variety, robustness, or unpredictability in a specified, controlled manner. This framework, also termed as *control improvisation* or *controlled improvisation*, was proposed and formalized by the author and colleagues several years ago [2, 6, 5]. Informally, an improviser is a generator of data items  $d_1, d_2, d_3, \dots$  subject to three kinds of constraints:

1. *Hard Constraints*: Each data item  $d_i$  must satisfy all these constraints.
2. *Soft Constraints*: A data item  $d_i$  must satisfy these constraints as measured by a tunable quantity, typically a probability written as  $1 - \delta$  for tunable parameter  $\delta$ .



© Sanjit A. Seshia;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 5; pp. 5:1–5:3

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3. *Randomness Constraints*: The output distribution of the improviser must satisfy specified properties, e.g., obeying a particular distribution.

The problem of synthesizing an improviser is termed as the control (or controlled) improvisation problem. The papers and thesis by Fremont et al. [6, 5, 4] lay out the foundations of the theory of control improvisation, analyzing its complexity for different variants of the problem involving various forms of constraints.

Algorithmic improvisation has been demonstrated in a variety of applications. Here are some of these applications:

- Music improvisation, generating controlled random variations of a given melody [2];
- Modeling human behavior for controlling Internet-of-Things (IoT) devices in a home automation context [1];
- Synthesizing control policies for controlling vehicles [9];
- Synthesizing randomized plans for robotic surveillance [3];
- Generating test inputs for software fuzz testing [4];
- Generating test cases for simulating cyber-physical systems [8, 7], and
- Generation of synthetic data for training and testing machine learning applications [7].

In these applications, the type of data generated by the improviser varies (music, control policies, test inputs, images, etc.) and the formalism used to encode constraints also varies, including logics, automata, and domain specific languages.

In this invited talk, I will give an overview of the theory of algorithmic improvisation, give a tour of some of the key applications with a particular focus on the design of intelligent autonomous systems, and present an outlook on the exciting future directions that remain to be explored.

---

## References

- 1 Ilge Akkaya, Daniel Fremont, Rafael Valle, Alexandre Donz , Edward A. Lee, and Sanjit A. Seshia. Control improvisation for probabilistic temporal specifications. In *Proceedings of the 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2016.
- 2 Alexandre Donz , Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, pages 1277–1284, September 2014.
- 3 Daniel Fremont and Sanjit A. Seshia. Reactive control improvisation. In *30th International Conference on Computer Aided Verification (CAV)*, 2018.
- 4 Daniel J. Fremont. *Algorithmic Improvisation*. PhD thesis, EECS Department, University of California, Berkeley, August 2019. URL: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-133.html>.
- 5 Daniel J. Fremont, Alexandre Donz , and Sanjit A. Seshia. Control improvisation. *CoRR*, abs/1704.06319, 2017.
- 6 Daniel J. Fremont, Alexandre Donz , Sanjit A. Seshia, and David Wessel. Control improvisation. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 45 of *LIPICs*, pages 463–474, December 2015.
- 7 Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2019.
- 8 Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Brusio, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In *23rd IEEE International Conference on Intelligent Transportation Systems (ITSC)*, September 2020.

- 9 Jin I. Ge and Richard M. Murray. Voluntary lane-change policy synthesis with control improvisation. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 3640–3647. IEEE, 2018.



# On Some Recent Advances in Algebraic Complexity

Amir Shpilka 

School of Computer Science, Tel Aviv University, Israel

<https://www.cs.tau.ac.il/~shpilka>

shpilka@tauex.tau.ac.il

---

## Abstract

Algebraic complexity is the field studying the intrinsic difficulty of algebraic problems in an algebraic model of computation, most notably arithmetic circuits. It is a very natural model of computation that attracted a large amount of research in the last few decades, partially due to its simplicity and elegance, but mostly because of its importance. Being a more structured model than Boolean circuits, one could hope that the fundamental problems of theoretical computer science, such as separating P from NP, deciding whether  $P = BPP$  and more, will be easier to solve for arithmetic circuits.

In this talk I will give the basic definitions, explain the main questions and how they relate to their Boolean counterparts, and discuss what I view as promising approaches to tackling the most fundamental problems in the field.

**2012 ACM Subject Classification** Theory of computation → Algebraic complexity theory

**Keywords and phrases** Algebraic Complexity, Arithmetic Circuits, Polynomial Identity Testing

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.6

**Category** Invited Talk

**Funding** *Amir Shpilka*: Supported by the Israel Science Foundation (grant number 514/20) and by the Len Blavatnik and the Blavatnik Family foundation-



© Amir Shpilka;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 6; pp. 6:1–6:1



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Faster Property Testers in a Variation of the Bounded Degree Model

Isolde Adler 

University of Leeds, School of Computing, UK  
i.m.adler@leeds.ac.uk

Polly Fahey 

University of Leeds, School of Computing, UK  
mm11pf@leeds.ac.uk

---

## Abstract

---

Property testing algorithms are highly efficient algorithms, that come with probabilistic accuracy guarantees. For a property  $P$ , the goal is to distinguish inputs that have  $P$  from those that are *far* from having  $P$  with high probability correctly, by querying only a small number of local parts of the input. In property testing on graphs, the *distance* is measured by the number of edge modifications (additions or deletions), that are necessary to transform a graph into one with property  $P$ . Much research has focussed on the *query complexity* of such algorithms, i. e. the number of queries the algorithm makes to the input, but in view of applications, the *running time* of the algorithm is equally relevant.

In (Adler, Harwath STACS 2018), a natural extension of the bounded degree graph model of property testing to relational databases of bounded degree was introduced, and it was shown that on databases of bounded degree and bounded tree-width, every property that is expressible in monadic second-order logic with counting (CMSO) is testable with constant query complexity and *sublinear* running time. It remains open whether this can be improved to constant running time.

In this paper we introduce a new model, which is based on the bounded degree model, but the distance measure allows both edge (tuple) modifications and vertex (element) modifications. Our main theorem shows that on databases of bounded degree and bounded tree-width, every property that is expressible in CMSO is testable with constant query complexity and *constant* running time in the new model. We also show that every property that is testable in the classical model is testable in our model with the same query complexity and running time, but the converse is not true.

We argue that our model is natural and our meta-theorem showing constant-time CMSO testability supports this.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Database query processing and optimization (theory)

**Keywords and phrases** Constant Time Algorithms, Logic and Databases, Property Testing, Bounded Degree Model

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.7

## 1 Introduction

Extracting information from large amounts of data and understanding its global structure can be an immensely challenging and time consuming task. When the input data is huge, many traditionally “efficient” algorithms are no longer practical. The framework of property testing aims at addressing this problem. Property testing algorithms (*testers*, for short) are given oracle access to the inputs, and their goal is to distinguish between inputs which have a given property  $\mathbf{P}$  or are structurally *far* from having  $\mathbf{P}$  with high probability correctly. This can be seen as a relaxation of the classical yes/no decision problem for  $\mathbf{P}$ . Testers make these decisions by exploring only a small number of local parts of the input which are randomly chosen. They come with probabilistic guarantees on the quality of the answer.



© Isolde Adler and Polly Fahey;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 7; pp. 7:1–7:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Typically, only a constant number of small local parts are explored and the algorithms often run in constant or sublinear time. This speed up in running time, whilst sacrificing some accuracy, can be crucial for dealing with large inputs. In particular it can be useful for a quick exploration of newly obtained data (e.g. biological networks). Based on the outcome of the exploration, a decision can then be taken whether to use a more time consuming exact algorithm in a second step.

A *property* is simply an isomorphism-closed class of graphs or relational databases. For example, each Boolean database query  $q$  defines a property  $\mathbf{P}_q$ , the class of all databases satisfying  $q$ . In the bounded degree graph model [16], a uniform upper bound  $d$  on the degree of the graphs is assumed. For a small  $\epsilon \in (0, 1]$ , two graphs  $\mathcal{G}$  and  $\mathcal{H}$ , both on  $n$  vertices, are  $\epsilon$ -close, if at most  $\epsilon dn$  edge modifications (deletions or insertions in  $\mathcal{G}$  or  $\mathcal{H}$ ) are necessary to make  $\mathcal{G}$  and  $\mathcal{H}$  isomorphic. If  $\mathcal{G}$  and  $\mathcal{H}$  are not  $\epsilon$ -close, then they are called  $\epsilon$ -far. A graph  $\mathcal{G}$  is called  $\epsilon$ -close to a property  $\mathbf{P}$ , if  $\mathcal{G}$  is  $\epsilon$ -close to a member of  $\mathbf{P}$ , and  $\mathcal{G}$  is  $\epsilon$ -far from  $\mathbf{P}$  otherwise. The natural generalisation of this model to relational databases of bounded degree (where a database has degree at most  $d$  if each element in its domain appears in at most  $d$  tuples) was studied in [1], where two databases  $\mathcal{D}$  and  $\mathcal{D}'$ , both with  $n$  elements in the domain, are  $\epsilon$ -close, if at most  $\epsilon dn$  tuple modifications (deletions from relations or insertions to relations) are necessary to make  $\mathcal{D}$  and  $\mathcal{D}'$  isomorphic, and  $\mathcal{D}$  and  $\mathcal{D}'$  are  $\epsilon$ -far otherwise. We call this model for bounded degree relational databases the BDRD model.

**Our contributions.** In this paper we propose a new model for property testing on bounded degree relational databases, which we call the  $\text{BDRD}_{+/-}$  model, with a distance measure that allows both tuple deletions and insertions, and *deletion and insertion of elements of the domain*. On graphs, this translates to edge insertions and deletions, and *vertex insertions and deletions*. We argue that this yields a natural distance measure. Indeed, take any (sufficiently large) graph  $\mathcal{G}$ , and let  $\mathcal{H}$  be obtained from  $\mathcal{G}$  by adding an isolated vertex. Then  $\mathcal{G}$  and  $\mathcal{H}$  are  $\epsilon$ -far for every  $\epsilon \in (0, 1]$  under the classical distance measure, although they only differ in one vertex. In contrast, our distance measure allows for a small number of vertex modifications. While comparing graphs on different numbers of vertices by adding isolated vertices was done implicitly as part of the study the testability of outerplanar graphs [4], to the best of our knowledge, such a distance measure has not been considered before as part of a model in property testing, which seems surprising to us.

Formally, in the  $\text{BDRD}_{+/-}$  model, two databases  $\mathcal{D}$  and  $\mathcal{D}'$  are  $\epsilon$ -close, if they can be made isomorphic by at most  $\epsilon dn$  modifications, where a modification is either, (1) removing a tuple from a relation, (2) inserting a tuple to a relation, (3) removing an element from the domain (and, as a consequence, any tuple containing that element is removed), or (4) inserting an element into the domain. Here  $n$  is the minimum of the sizes of the domains of  $\mathcal{D}$  and  $\mathcal{D}'$ . In Section 3 we give the full details of our model. We note that the  $\text{BDRD}_{+/-}$  model differs from the BDRD model only in the choice of the distance measure. While we work in the setting of relational databases, we would like to emphasize that our results carry over to (undirected and directed) graphs, as these can be seen as special instances of relational databases.

It is known that in the bounded degree graph model, every minor-closed property is testable [6], and, more generally, every hyperfinite graph property is testable [23] with constant query complexity. However, no bound on the running time can be obtained in these general settings. Indeed, there exist hyperfinite properties (of edgeless graphs) that are uncomputable. In [1], Adler and Harwath ask which conditions guarantee both low query complexity *and* efficient running time. They prove a meta-theorem stating that, on classes of

databases (or graphs) of bounded degree and bounded tree-width, every property that can be expressed by a sentence of monadic second-order logic with counting (CMSO) is testable with *constant* query complexity and *polylogarithmic* running time in the BDRD model. Treating many algorithmic problems simultaneously, this can be seen as an algorithmic *meta-theorem* within the line of research inspired by Courcelle’s famous theorem [9] that states that each property of relational databases which is definable in CMSO is decidable in linear time on relational databases of bounded tree-width. CMSO extends first-order logic (FO) and hence properties expressible in FO (e.g. subgraph/sub-database freeness) are also expressible in CMSO. Other examples of graph properties expressible in CMSO include bipartiteness, colourability, even-hole-freeness and Hamiltonicity. Rigidity (i. e. the absence of a non-trivial automorphism) cannot be expressed in CMSO (cf. [10] for more details).

Our main theorem (Theorem 17) shows that in the  $\text{BDRD}_{+/-}$  model, on classes of databases (or graphs) of bounded degree and bounded tree-width, every property that can be expressed by a sentence of monadic second-order logic with counting (CMSO) is testable with *constant* query complexity and *constant* running time. The question whether constant running time can also be achieved in the BDRD model remains open.

We show that the  $\text{BDRD}_{+/-}$  model is in fact stronger than the BDRD model: Any property testable in the BDRD model is also testable in the  $\text{BDRD}_{+/-}$  model with the same query complexity and running time (Lemma 4), but there are examples that show that the converse is not true (Lemma 6).

In the future, it would be interesting to obtain a characterisation of the properties that are (efficiently) testable in the  $\text{BDRD}_{+/-}$  model.

**Our techniques.** To prove our main theorem, we give a general condition under which properties are testable in constant time in the  $\text{BDRD}_{+/-}$  model whereas the fastest known testers for such properties in the BDRD model run in polylogarithmic time. To describe this condition let us first briefly introduce some definitions. A property  $\mathbf{P}$  is *hyperfinite* on a class of databases  $\mathbf{C}$  if every database in  $\mathbf{P}$  can be partitioned into connected components of constant size by removing only a constant fraction of the tuples such that the resulting partitioned database is in  $\mathbf{C}$ . Let  $r \in \mathbb{N}$ , given an element  $a$  in the domain of a database  $\mathcal{D}$  the  *$r$ -neighbourhood type* of  $a$  in  $\mathcal{D}$  is the isomorphism type of the sub-database of  $\mathcal{D}$  induced by all elements that are at distance at most  $r$  from  $a$  in the underlying graph of  $\mathcal{D}$ , expanded by  $a$ . The  *$r$ -histogram* of a bounded degree database  $\mathcal{D}$ , denoted by  $h_r(\mathcal{D})$ , is a vector indexed by the  $r$ -neighbourhood types, where the component corresponding to the  $r$ -neighbourhood type  $\tau$  contains the number of elements in  $\mathcal{D}$  that realise  $\tau$ . The  *$r$ -neighbourhood distribution* of  $\mathcal{D}$  is the vector  $h_r(\mathcal{D})/n$  where  $\mathcal{D}$  is on  $n$  elements. We show that for any property  $\mathbf{P}$  and input class  $\mathbf{C}$ , if  $\mathbf{P}$  is hyperfinite on  $\mathbf{C}$  and the set of  $r$ -histograms of the databases in  $\mathbf{P}$  are semilinear, then  $\mathbf{P}$  is testable on  $\mathbf{C}$  in constant time (Theorem 16). As a corollary we then obtain our main theorem, that every property definable by a CMSO sentence is testable on the class of databases with bounded degree and bounded tree-width in constant time (Theorem 17).

Alon [22, Proposition 19.10] proved that for every bounded degree graph  $\mathcal{G}$  there exists a constant size graph  $\mathcal{H}$  that has a similar neighbourhood distribution to  $\mathcal{G}$ . However, the proof is based on a compactness argument and does not give an explicit upper bound on the size of  $\mathcal{H}$ . Finding such a bound was suggested by Alon as an open problem [18]. We ask under which conditions on a given property  $\mathbf{P}$ , for every member of  $\mathbf{P}$  there exists a constant size database with a similar neighbourhood distribution which is also in  $\mathbf{P}$ . We show that for any property  $\mathbf{P}$  which is hyperfinite on the input class  $\mathbf{C}$  and whose  $r$ -histograms are

semilinear, if a database  $\mathcal{D}$  is in  $\mathbf{P}$  then there exists a constant size database  $\mathcal{D}'$  in  $\mathbf{P}$  with a similar neighbourhood distribution but this is not true for databases in  $\mathbf{C}$  that are far from  $\mathbf{P}$ . Furthermore, we obtain upper and lower bounds on the size of  $\mathcal{D}'$ . We can then use this result to construct constant time testers. We first use the algorithm `EstimateFrequencies $r,s$`  (given in [23] and adapted to databases in [1]) to approximate the neighbourhood distribution of the input database. Then we only have to check if the estimated distribution is close to the neighbourhood distribution of a constant size database in the property.

As a corollary (Corollary 14), we obtain an explicit bound on the size on graphs  $\mathcal{H}$  from Alon’s theorem for “semilinear” properties, i. e. properties, where the histogram vectors of the neighbourhood distributions form a semilinear set.

**Further related work.** Other than the work already mentioned in [1] there are only a handful of results on relational databases that utilise models from property testing. Chen and Yoshida [8] study a model which is close to the general graph model (cf. e. g. [2]) in which they study the testability of homomorphism inadmissibility. Ben-Moshe et al. [5] study the testability of near-sortedness (a property of relations that states that most tuples are close to their place in some desired order). Our model differs from both of these, as it relies on a degree bound and uses different types of oracle access. Explicit bounds for Alon’s theorem restricted to high-girth graphs were given in [12].

Obtaining a characterisation of constant query testable properties is a long-standing open problem. Ito et al. [19] give a characterisation of the 1-sided error constant query testable monotone and hereditary graph properties in the bounded degree (directed and undirected) graph model. Fichtenberger et al. [13] show that every constant query testable property in the bounded degree graph model is either finite or contains an infinite hyperfinite subproperty.

**Organisation.** In Section 2 we introduce relevant notions used throughout the paper. In Section 3 we introduce our property testing model for bounded degree relational databases and we compare it to the classical model. In Section 4 we prove our main theorems. Due to space constraints the proofs of statements labelled (\*) are deferred to the appendix.

## 2 Preliminaries

We let  $\mathbb{N}$  be the set of natural numbers including 0, and  $\mathbb{N}_{\geq 1} = \mathbb{N} \setminus \{0\}$ . For each  $n \in \mathbb{N}_{\geq 1}$ , we let  $[n] = \{1, 2, \dots, n\}$ .

**Databases.** A *schema* is a finite set  $\sigma = \{R_1, \dots, R_{|\sigma|}\}$  of relation names, where each  $R \in \sigma$  has an *arity*  $\text{ar}(R) \in \mathbb{N}_{\geq 1}$ . A *database*  $\mathcal{D}$  of schema  $\sigma$  ( $\sigma$ -db for short) is of the form  $\mathcal{D} = (D, R_1^{\mathcal{D}}, \dots, R_{|\sigma|}^{\mathcal{D}})$ , where  $D$  is a finite set, the set of *elements* of  $\mathcal{D}$ , and  $R_i^{\mathcal{D}}$  is an  $\text{ar}(R_i)$ -ary relation on  $D$ . The set  $D$  is also called the *domain* of  $\mathcal{D}$ . An (*undirected*) *graph*  $\mathcal{G}$  is a tuple  $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}))$  where  $V(\mathcal{G})$  is a set of *vertices* and  $E(\mathcal{G})$  is a set of 2-element subsets of  $V(\mathcal{G})$  (the *edges* of  $\mathcal{G}$ ). An undirected graph can be seen as a  $\{E\}$ -db, where  $E$  is a binary relation name, interpreted by a symmetric, irreflexive relation.

We assume that all databases are linearly ordered or, equivalently, that  $D = [n]$  for some  $n \in \mathbb{N}$  (similar to [20]). We extend this linear ordering to a linear order on the relations of  $\mathcal{D}$  via lexicographic ordering. The *Gaiifman graph* of a  $\sigma$ -db  $\mathcal{D}$  is the undirected graph  $\mathcal{G}(\mathcal{D}) = (V, E)$ , with vertex set  $V := D$  and an edge between vertices  $a$  and  $b$  whenever  $a \neq b$  and there is an  $R \in \sigma$  and a tuple  $(a_1, \dots, a_{\text{ar}(R)}) \in R^{\mathcal{D}}$  with  $a, b \in \{a_1, \dots, a_{\text{ar}(R)}\}$ . The

*degree*  $\deg(a)$  of an element  $a$  in a database  $\mathcal{D}$  is the total number of tuples in all relations of  $\mathcal{D}$  that contain  $a$ . We say the *degree*  $\deg(\mathcal{D})$  of a database  $\mathcal{D}$  is the maximum degree of its elements. A class of databases  $\mathbf{C}$  has *bounded degree*, if there exists a constant  $d \in \mathbb{N}$  such that for all  $\mathcal{D} \in \mathbf{C}$ ,  $\deg(\mathcal{D}) \leq d$ . (We always assume that classes of databases are closed under isomorphism.) Let us remark that the  $\deg(\mathcal{D})$  and the (graph-theoretic) degree of  $\mathcal{G}(\mathcal{D})$  only differ by at most a constant factor (cf. e.g. [11]). Hence both measures yield the same classes of relational structures of bounded degree. We define the *tree-width* of a database  $\mathcal{D}$  as the tree-width of its Gaifman graph. (See e.g. [15] for a discussion of tree-width in this context.) A class  $\mathbf{C}$  of databases has *bounded tree-width*, if there exists a constant  $t \in \mathbb{N}$  such that all databases  $\mathcal{D} \in \mathbf{C}$  have tree-width at most  $t$ . Let  $\mathcal{D}$  be a  $\sigma$ -db, and  $M \subseteq D$ . The sub-database of  $\mathcal{D}$  *induced by*  $M$  is the database  $\mathcal{D}[M]$  with domain  $M$  and  $R^{\mathcal{D}[M]} := R^{\mathcal{D}} \cap M^{\text{ar}(R)}$  for every  $R \in \sigma$ . An  $(\epsilon, k)$ -*partition* of a  $\sigma$ -db  $\mathcal{D}$  on  $n$  elements is a  $\sigma$ -db  $\mathcal{D}'$  formed by removing at most  $\epsilon n$  many tuples from  $\mathcal{D}$  such that every connected component in  $\mathcal{D}'$  contains at most  $k$  elements. A class of  $\sigma$ -dbs  $\mathbf{C} \subseteq \mathbf{D}$  is  $\rho$ -*hyperfinite* on  $\mathbf{D}$  if for every  $\epsilon \in (0, 1]$  and  $\mathcal{D} \in \mathbf{C}$  there exists an  $(\epsilon, \rho(\epsilon))$ -partition  $\mathcal{D}' \in \mathbf{D}$  of  $\mathcal{D}$ . We call  $\mathbf{C}$  *hyperfinite* on  $\mathbf{D}$  if there exists a function  $\rho$  such that  $\mathbf{C}$  is  $\rho$ -hyperfinite on  $\mathbf{D}$ .

**Logics.** We shall only briefly introduce first-order logic (FO) and monadic second-order logic with counting (CMSO). Detailed introductions can be found in [21] and [10]. Let  $\mathbf{var}$  be a countable infinite set of *variables*, and fix a relational schema  $\sigma$ . The set  $\text{FO}[\sigma]$  is built from *atomic formulas* of the form  $x_1 = x_2$  or  $R(x_1, \dots, x_{\text{ar}(R)})$ , where  $R \in \sigma$  and  $x_1, \dots, x_{\text{ar}(R)} \in \mathbf{var}$ , and is closed under Boolean connectives ( $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ ) and existential and universal quantifications ( $\exists, \forall$ ). *Monadic second-order logic* (MSO) is the extension of first-order logic that also allows quantification over subsets of the domain. CMSO extends MSO by allowing first-order modular counting quantifiers  $\exists^m$  for every integer  $m$  (where  $\exists^m \phi$  is true in a  $\sigma$ -db if the number of its elements for which  $\phi$  is satisfied is divisible by  $m$ ). A *free variable* of a formula is a (individual or set) variable that does not appear in the scope of a quantifier. A formula without free variables is called a *sentence*. For a  $\sigma$ -db  $\mathcal{D}$  and a sentence  $\phi$  we write  $\mathcal{D} \models \phi$  to denote that  $\mathcal{D}$  satisfies  $\phi$ .

► **Proviso.** *For the rest of the paper, we fix a schema  $\sigma$  and numbers  $d, t \in \mathbb{N}$  with  $d \geq 2$ . From now on, all databases are  $\sigma$ -dbs and have degree at most  $d$ , unless stated otherwise. We use  $\mathbf{C}_d$  to denote the class of all  $\sigma$ -dbs with degree at most  $d$ ,  $\mathbf{C}_d^t$  to denote the class of all  $\sigma$ -dbs with degree at most  $d$  and tree-width at most  $t$  and finally we use  $\mathbf{C}$  to denote a class of  $\sigma$ -dbs with degree at most  $d$ .*

**Property testing.** Adler and Harwath [1] introduced the model of property testing for bounded degree relational databases, which is a straightforward extension of the model for bounded degree graphs [16]. We call this model the BDRD *model* for short, which we shall discuss below.

Property testing algorithms do not have access to the whole input database. Instead, they are given access via an *oracle*. Let  $\mathcal{D}$  be an input  $\sigma$ -db on  $n$  elements. A property testing algorithm receives the number  $n$  as input, and it can make *oracle queries*<sup>1</sup> of the form  $(R, i, j)$ , where  $R \in \sigma$ ,  $i \leq n$  and  $j \leq \deg(\mathcal{D})$ . The answer to  $(R, i, j)$  is the  $j^{\text{th}}$  tuple in  $R^{\mathcal{D}}$  containing the  $i^{\text{th}}$  element<sup>2</sup> of  $\mathcal{D}$  (if such a tuple does not exist then it returns  $\perp$ ). We assume oracle queries are answered in constant time.

<sup>1</sup> Note that an oracle query is not a database query.

<sup>2</sup> According to the assumed linear order on  $D$ .

Let  $\mathcal{D}, \mathcal{D}'$  be two  $\sigma$ -dbs, both having  $n$  elements. In the BDRD model the *distance* between  $\mathcal{D}$  and  $\mathcal{D}'$ , denoted by  $\text{dist}(\mathcal{D}, \mathcal{D}')$ , is the minimum number of tuples that have to be inserted or removed from relations of  $\mathcal{D}$  and  $\mathcal{D}'$  to make  $\mathcal{D}$  and  $\mathcal{D}'$  isomorphic. For  $\epsilon \in [0, 1]$ , we say  $\mathcal{D}$  and  $\mathcal{D}'$  are  $\epsilon$ -close if  $\text{dist}(\mathcal{D}, \mathcal{D}') \leq \epsilon dn$ , and  $\mathcal{D}$  and  $\mathcal{D}'$  are  $\epsilon$ -far otherwise. A *property* is simply an isomorphism-closed class of databases. Note that every CMSO sentence  $\phi$  defines a property  $\mathbf{P}_\phi = \{\mathcal{D} \mid \mathcal{D} \models \phi\}$ . We call  $\mathbf{P}_\phi \cap \mathbf{C}$  the property *defined by  $\phi$  on  $\mathbf{C}$* . A  $\sigma$ -db  $\mathcal{D}$  is  $\epsilon$ -close to a property  $\mathbf{P}$  if there exists a database  $\mathcal{D}' \in \mathbf{P}$  that is  $\epsilon$ -close to  $\mathcal{D}$ , otherwise  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$ .

Let  $\mathbf{P} \subseteq \mathbf{C}$  be a property and  $\epsilon \in (0, 1]$  be the proximity parameter. An  $\epsilon$ -tester for  $\mathbf{P}$  on  $\mathbf{C}$  is a probabilistic algorithm which is given oracle access to a  $\sigma$ -db  $\mathcal{D} \in \mathbf{C}$  and it is given  $n := |\mathcal{D}|$  as auxiliary input. The algorithm does the following:

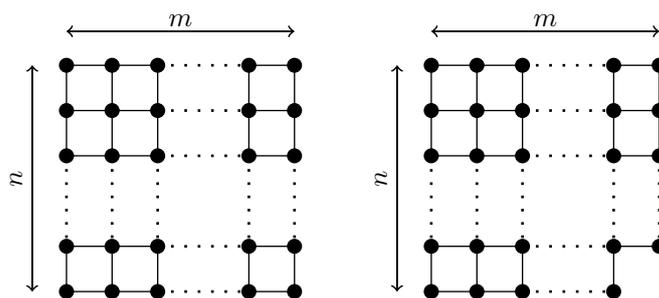
1. If  $\mathcal{D} \in \mathbf{P}$ , then the tester accepts with probability at least  $2/3$ .
2. If  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$ , then the tester rejects with probability at least  $2/3$ .

The *query complexity* of a tester is the maximum number of oracle queries made. A tester has *constant* query complexity, if the query complexity does not depend on the size of the input database. We say a property  $\mathbf{P} \subseteq \mathbf{C}$  is *uniformly testable* in time  $f(n)$  on  $\mathbf{C}$ , if for every  $\epsilon \in (0, 1]$  there exists an  $\epsilon$ -tester for  $\mathbf{P}$  on  $\mathbf{C}$  which has constant query complexity and whose running time on databases on  $n$  elements is  $f(n)$ . Note that this tester must work for all  $n$ .

**Neighbourhoods.** For a  $\sigma$ -db  $\mathcal{D}$  and  $a, b \in D$ , the *distance* between  $a$  and  $b$  in  $\mathcal{D}$ , denoted by  $\text{dist}_{\mathcal{D}}(a, b)$ , is the length of a shortest path between  $a$  and  $b$  in  $\mathcal{G}(\mathcal{D})$ . Let  $r \in \mathbb{N}$ . For an element  $a \in D$ , we let  $N_r^{\mathcal{D}}(a)$  denote the set of all elements of  $\mathcal{D}$  that are at distance at most  $r$  from  $a$ . The  $r$ -neighbourhood of  $a$  in  $\mathcal{D}$ , denoted by  $\mathcal{N}_r^{\mathcal{D}}(a)$ , is the tuple  $(\mathcal{D}[N_r^{\mathcal{D}}(a)], a)$  where  $a$  is called the *centre*. We omit the superscript and write  $N_r(a)$  and  $\mathcal{N}_r(a)$ , if  $\mathcal{D}$  is clear from the context. Two  $r$ -neighbourhoods,  $\mathcal{N}_r(a)$  and  $\mathcal{N}_r(b)$ , are *isomorphic* (written  $\mathcal{N}_r(a) \cong \mathcal{N}_r(b)$ ) if there is an isomorphism between  $\mathcal{D}[N_r(a)]$  and  $\mathcal{D}[N_r(b)]$  which maps  $a$  to  $b$ . An  $\cong$ -equivalence-class of  $r$ -neighbourhoods is called an  $r$ -neighbourhood type (or  $r$ -type for short). We let  $T_r^{\sigma, d}$  denote the set of all  $r$ -types with degree at most  $d$ , over schema  $\sigma$ . Note that for fixed  $d$  and  $\sigma$ , the cardinality  $|T_r^{\sigma, d}| =: c(r)$  is a constant, only depending on  $r$  and  $d$ . We say that an element  $a \in D$  has  $r$ -type  $\tau$ , if  $\mathcal{N}_r^{\mathcal{D}}(a) \in \tau$ . For  $r \in \mathbb{N}$ , the  $r$ -histogram of a database  $\mathcal{D}$ , denoted by  $h_r(\mathcal{D})$ , is the vector with  $c(r)$  components, indexed by the  $r$ -types, where the component corresponding to type  $\tau$  contains the number of elements of  $\mathcal{D}$  of  $r$ -type  $\tau$ . The  $r$ -neighbourhood distribution of  $\mathcal{D}$ , denoted by  $\text{dv}_r(\mathcal{D})$ , is the vector  $h_r(\mathcal{D})/n$  where  $|D| = n$ . For a class of  $\sigma$ -dbs  $\mathbf{C}$  and  $r \in \mathbb{N}$ , we let  $h_r(\mathbf{C}) := \{h_r(\mathcal{D}) \mid \mathcal{D} \in \mathbf{C}\}$ . A set is *semilinear* if it is a finite union of linear sets. A set  $M \subseteq \mathbb{N}^c$  is linear if  $M = \{\bar{v}_0 + a_1 \bar{v}_1 + \dots + a_k \bar{v}_k \mid a_1, \dots, a_k \in \mathbb{N}\}$ , for some  $\bar{v}_0, \dots, \bar{v}_k \in \mathbb{N}^c$ . From a result in [14] about many-sorted spectra of CMSO sentences it can be derived that that the set of  $r$ -histograms of properties defined by a CMSO sentence on  $\mathbf{C}_d^t$  are semilinear.

► **Lemma 1** ([1, 14]). *For each  $r \in \mathbb{N}$  and each property  $\mathbf{P} \subseteq \mathbf{C}_d^t$  definable by a CMSO sentence on  $\mathbf{C}_d^t$ , the set  $h_r(\mathbf{P})$  is semilinear.*

**Model of computation.** We use Random Access Machines (RAMs) and a uniform cost measure when analysing our algorithms, i.e. we assume all basic arithmetic operations including random sampling can be done in constant time, regardless of the size of the numbers involved.



■ **Figure 1** The graphs  $\mathcal{G}_{n,m}$  and  $\mathcal{H}_{n,m}$  (respectively) of Example 3.

### 3 The Model

We shall now introduce our property testing model for bounded degree relational databases, which is an extension of the BDRD model discussed in Section 2. The notions of oracle queries, properties,  $\epsilon$ -tester, query complexity and uniform testability remain the same but we have an alternative definition of distance and  $\epsilon$ -closeness. In our model, which we shall call the BDRD $_{+/-}$  model for short, we can add and remove elements as well as tuples and can therefore compare databases that are on a different number of elements.

► **Definition 2** (Distance and  $\epsilon$ -closeness). *Let  $\mathcal{D}, \mathcal{D}' \in \mathbf{C}_d$  and  $\epsilon \in [0, 1]$ . The distance between  $\mathcal{D}$  and  $\mathcal{D}'$  (denoted by  $\text{dist}_{+/-}(\mathcal{D}, \mathcal{D}')$ ) is the minimum number of modifications we need to make to  $\mathcal{D}$  and  $\mathcal{D}'$  to make them isomorphic where a modification is either (1) inserting a new element, (2) deleting an element (and as a result deleting any tuple that contains that element), (3) inserting a tuple, or (4) deleting a tuple. We then say  $\mathcal{D}$  and  $\mathcal{D}'$  are  $\epsilon$ -close if  $\text{dist}_{+/-}(\mathcal{D}, \mathcal{D}') \leq \epsilon d \min\{|\mathcal{D}|, |\mathcal{D}'|\}$  and are  $\epsilon$ -far otherwise.*

The following example illustrates the difference between the distance measure of the BDRD and the distance measure of the BDRD $_{+/-}$  model.

► **Example 3.** Let  $\mathbf{P} = \{\mathcal{G}_{n,m} \mid n, m \in \mathbb{N}_{>1}\}$  where  $\mathcal{G}_{n,m}$  is an  $n$  by  $m$  grid graph as shown in Figure 1. Let us consider the graph  $\mathcal{H}_{n,m}$  for some  $n, m \in \mathbb{N}$  which is formed from  $\mathcal{G}_{n,m}$  by removing a corner vertex. In the BDRD $_{+/-}$  model the distance between  $\mathcal{H}_{n,m}$  and  $\mathcal{G}_{n,m}$  is 1 (we remove a corner vertex from  $\mathcal{G}_{n,m}$  to get  $\mathcal{H}_{n,m}$ ) and therefore  $\mathcal{H}_{n,m}$  is at distance 1 from  $\mathbf{P}$  in the BDRD $_{+/-}$  model. In the BDRD model if two graphs are on a different number of vertices then the distance between them is infinity. Therefore if  $nm - 1$  is a prime number then  $\mathcal{H}_{n,m}$  is at distance infinity from  $\mathbf{P}$  in the BDRD model.

We now show that if a property is testable in the BDRD model then it is also testable in the BDRD $_{+/-}$  model but the converse is not true. This allows for more testable properties in the BDRD $_{+/-}$  model.

► **Lemma 4** (\*). *Let  $\mathbf{P} \subseteq \mathbf{C}$ . If  $\mathbf{P}$  is uniformly testable on  $\mathbf{C}$  in time  $f(n)$  in the BDRD model then  $\mathbf{P}$  is also uniformly testable on  $\mathbf{C}$  in time  $f(n)$  in the BDRD $_{+/-}$  model.*

► **Theorem 5** ([16]). *In the bounded degree model, bipartiteness cannot be tested with query complexity  $o(\sqrt{n})$ , where  $n$  is the number of vertices of the input graph.*

► **Lemma 6.** *There exists a class  $\mathbf{C}$  of  $\sigma$ -dbs and a property  $\mathbf{P} \subseteq \mathbf{C}$  such that  $\mathbf{P}$  is trivially testable on  $\mathbf{C}$  in the BDRD $_{+/-}$  model but is not testable on  $\mathbf{C}$  in the BDRD model.*

**Proof.** Let  $\mathbf{C}$  be the class of all graphs with degree at most  $d$ . Let  $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2 \subseteq \mathbf{C}$  be the property where  $\mathbf{P}_1$  contains all bipartite graphs in  $\mathbf{C}$  and  $\mathbf{P}_2$  contains all graphs in  $\mathbf{C}$  that have an odd number of vertices. In the  $\text{BDRD}_{+/-}$  model every  $\mathcal{G} \in \mathbf{C}$  is  $\epsilon$ -close to  $\mathbf{P}$  if  $|V(\mathcal{G})| \geq 1/(\epsilon d)$  and hence  $\mathbf{P}$  is trivially testable on  $\mathbf{C}$  in the  $\text{BDRD}_{+/-}$  model (the tester accepts if  $|V(\mathcal{G})| \geq 1/(\epsilon d)$  and does a full check of the input otherwise). In the BDRD model, if the input graph has an even number of vertices then it is far from  $\mathbf{P}_2$  and so we have to test for  $\mathbf{P}_1$ . By Theorem 5, bipartiteness is not testable (with constant query complexity) in the BDRD model. In particular, in the proof of Theorem 5, Goldreich and Ron show that for any even  $n$  there exists two families,  $\mathcal{G}_1 \subseteq \mathbf{C}$  and  $\mathcal{G}_2 \subseteq \mathbf{C}$ , of  $n$ -vertex graphs such that every graph in  $\mathcal{G}_1$  is bipartite and almost all graphs in  $\mathcal{G}_2$  are far from being bipartite but any algorithm that performs  $o(\sqrt{n})$  queries cannot distinguish between a graph chosen randomly from  $\mathcal{G}_1$  and a graph chosen randomly from  $\mathcal{G}_2$ . Therefore  $\mathbf{P}$  is not testable on  $\mathbf{C}$  in the BDRD model.  $\blacktriangleleft$

Note that the underlying general principle of the above proof can be applied to obtain further examples of properties that are testable in the  $\text{BDRD}_{+/-}$  model but not testable in the BDRD model.

It is known that every hyperfinite property is “local” (Theorem 7), where “local” means that if a  $\sigma$ -db  $\mathcal{D}$  has a similar  $r$ -histogram to some  $\sigma$ -db (with the same domain size) that has the (hyperfinite) property, then  $\mathcal{D}$  must be  $\epsilon$ -close to the property [23, 1]. This is summarised in Theorem 7 below. We use Theorem 7 to prove a similar result in the  $\text{BDRD}_{+/-}$  model (Lemma 8). Lemma 8 is essential for the proof of Theorem 9.

► **Theorem 7** ([23, 1]). *Let  $\epsilon \in (0, 1]$  and let  $\mathbf{C}$  be closed under removing tuples. If a property  $\mathbf{P} \subseteq \mathbf{C}$  is hyperfinite on  $\mathbf{C}$  then there exists  $\lambda_7 := \lambda_7(\epsilon) \in (0, 1]$  and  $r_7 := r_7(\epsilon) \in \mathbb{N}$  such that for each  $\mathcal{D} \in \mathbf{P}$  and  $\mathcal{D}' \in \mathbf{C}$  with the same number  $n$  of elements, if  $\|h_{r_7}(\mathcal{D}) - h_{r_7}(\mathcal{D}')\|_1 \leq \lambda_7 n$ , then  $\mathcal{D}'$  is  $\epsilon$ -close to  $\mathbf{P}$  in the BDRD model.*

► **Lemma 8.** *Let  $\epsilon \in (0, 1]$  and let  $\mathbf{C}$  be closed under removing tuples. If a property  $\mathbf{P} \subseteq \mathbf{C}$  is hyperfinite on  $\mathbf{C}$  then there exists  $\lambda := \lambda(\epsilon) \in (0, 1]$  and  $r := r(\epsilon) \in \mathbb{N}$  such that for each  $\mathcal{D} \in \mathbf{P}$  and  $\mathcal{D}' \in \mathbf{C}$ , on  $|D|$  and  $|D'|$  elements respectively, if  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}')\|_1 \leq \lambda \min\{|D|, |D'|\}$ , then  $\mathcal{D}'$  is  $\epsilon$ -close to  $\mathbf{P}$  in the  $\text{BDRD}_{+/-}$  model.*

**Proof.** Let  $r = r_7(\epsilon/4)$  and let  $\lambda = \frac{\epsilon \lambda_7(\epsilon/4)}{1 + d^{r+1}}$ . Let us assume that  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}')\|_1 \leq \lambda \min\{|D|, |D'|\}$  and  $\mathbf{P}$  is hyperfinite on  $\mathbf{C}$ . If  $|D| = |D'|$  then by Theorem 7 and the choice of  $\lambda$ ,  $\mathcal{D}'$  is  $\epsilon$ -close to  $\mathbf{P}$ . So let us assume that  $|D| \neq |D'|$ . Let  $\mathcal{D}_1$  be the  $\sigma$ -db on  $|D|$  elements formed from  $\mathcal{D}'$  by either removing  $|D'| - |D|$  elements if  $|D| < |D'|$  or adding  $|D| - |D'|$  new elements if  $|D'| < |D|$ . Note that as  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}')\|_1 \leq \lambda \min\{|D|, |D'|\}$  and by definition  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}')\|_1 = \sum_{i=1}^{c(r)} |h_r(\mathcal{D}) - h_r(\mathcal{D}')|$  we have  $||D| - |D'|| \leq \lambda \min\{|D|, |D'|\}$ . When an element  $a$  is removed, the  $r$ -type of any element in  $N_r(a)$  will change. As  $|N_r(a)| \leq d^{r+1}$  (cf. e.g. Lemma 3.2 (a) of [7]) and  $||D| - |D'|| \leq \lambda \min\{|D|, |D'|\}$ , we have  $\|h_r(\mathcal{D}') - h_r(\mathcal{D}_1)\|_1 \leq \lambda \min\{|D|, |D'|\} d^{r+1}$ . Therefore

$$\|h_r(\mathcal{D}) - h_r(\mathcal{D}_1)\|_1 \leq \lambda \min\{|D|, |D'|\} (1 + d^{r+1}) \leq \lambda_7(\epsilon/4) |D|$$

by the choice of  $\lambda$ . By Theorem 7, in the BDRD model  $\mathcal{D}_1$  is  $\epsilon/4$ -close to  $\mathbf{P}$ . Hence there exists a  $\sigma$ -db  $\mathcal{D}_2 \in \mathbf{P}$  such that  $|D_2| = |D|$  and  $\text{dist}(\mathcal{D}_1, \mathcal{D}_2) \leq \epsilon d |D| / 4$ . By the definition of the two distance measures  $\text{dist}$  and  $\text{dist}_{+/-}$ , we have  $\text{dist}_{+/-}(\mathcal{D}_1, \mathcal{D}_2) \leq \text{dist}(\mathcal{D}_1, \mathcal{D}_2) \leq \epsilon d |D| / 4$  and by the choice of  $\mathcal{D}_1$  we have  $\text{dist}_{+/-}(\mathcal{D}', \mathcal{D}_1) \leq \lambda \min\{|D|, |D'|\}$ . Therefore

$$\text{dist}_{+/-}(\mathcal{D}', \mathcal{D}_2) \leq \frac{\epsilon d |D|}{4} + \lambda \min\{|D|, |D'|\} \leq \epsilon d \min\{|D|, |D'|\},$$

as  $|D| \leq \min\{|D|, |D'|\} + \lambda \min\{|D|, |D'|\} \leq 2 \min\{|D|, |D'|\}$  and  $\lambda \leq \epsilon d / 2$ . Hence in the  $\text{BDRD}_{+/-}$  model  $\mathcal{D}'$  is  $\epsilon$ -close to  $\mathbf{P}$ .  $\blacktriangleleft$

## 4 Main Results

We begin this section with the first of our main theorems (Theorem 9). We show that for any property  $\mathbf{P}$  which is hyperfinite on the input class  $\mathbf{C}$ , if the set of  $r$ -histograms of  $\mathbf{P}$  is semilinear, then for every  $\sigma$ -db  $\mathcal{D}$  in  $\mathbf{P}$  there exists a constant size  $\sigma$ -db in  $\mathbf{P}$  with a neighbourhood distribution similar to that of  $\mathcal{D}$ , but this is not true for  $\sigma$ -dbs in  $\mathbf{C}$  that are far from  $\mathbf{P}$ . We then use this result to prove that such properties are testable in constant time in the  $\text{BDRD}_{+/-}$  model (Theorem 16). As a corollary we obtain that CMSO definable properties on  $\sigma$ -dbs of bounded tree-width and bounded degree are testable in constant time (Theorem 17).

► **Theorem 9.** *Let  $\epsilon \in (0, 1]$  and let  $r := r(\epsilon)$  be as in Lemma 8. Let  $\mathbf{C}$  be closed under removing tuples and let  $\mathbf{P} \subseteq \mathbf{C}$  be a property that is hyperfinite on  $\mathbf{C}$  such that the set  $h_r(\mathbf{P})$  is semilinear. There exist  $n_{\min} := n_{\min}(\epsilon), n_{\max} := n_{\max}(\epsilon) \in \mathbb{N}$  and  $f := f(\epsilon), \mu := \mu(\epsilon) \in (0, 1)$  such that for every  $\mathcal{D} \in \mathbf{C}$  with  $|D| > n_{\max}$ ,*

1. *if  $\mathcal{D} \in \mathbf{P}$ , then there exists a  $\mathcal{D}' \in \mathbf{P}$  such that  $n_{\min} \leq |D'| \leq n_{\max}$  and  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 \leq f - \mu$ , and*
2. *if  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$  (in the  $\text{BDRD}_{+/-}$  model), then for every  $\mathcal{D}' \in \mathbf{P}$  such that  $n_{\min} \leq |D'| \leq n_{\max}$ , we have  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 > f + \mu$ .*

**Proof.** Let  $\lambda := \lambda(\epsilon)$  be as in Lemma 8 and  $c := c(r)$  (the number of  $r$ -types). First note that if  $\mathbf{P}$  is empty then for any choice of  $n_{\min}, n_{\max}, f$  and  $\mu$ , both 1. and 2. in the theorem statement are true and hence we shall assume that  $\mathbf{P}$  is non-empty. As  $h_r(\mathbf{P})$  is a semilinear set we can write it as follows,  $h_r(\mathbf{P}) = M_1 \cup M_2 \cup \dots \cup M_m$  where  $m \in \mathbb{N}$  and for each  $i \in [m]$ ,  $M_i = \{\bar{v}_0^i + a_1 \bar{v}_1^i + \dots + a_{k_i} \bar{v}_{k_i}^i \mid a_1, \dots, a_{k_i} \in \mathbb{N}\}$  is a linear set where  $\bar{v}_0^i, \dots, \bar{v}_{k_i}^i \in \mathbb{N}^c$  and for each  $j \in [k_i]$ ,  $\|\bar{v}_j^i\|_1 \neq 0$ . Let  $k := \max_{i \in [m]} k_i + 1$  and  $v := \max_{i \in [m]} \left( \max_{j \in [0, k_i]} \|\bar{v}_j^i\|_1 \right)$  (note that  $v > 0$  as  $\mathbf{P}$  is non-empty). Let  $n_{\min} := n_0 - kv, n_{\max} := n_0 + kv, f := \frac{\lambda}{3c}$ , and  $\mu := \frac{\lambda}{6c}$  where

$$n_0 := kv \left( \frac{3ckv}{f - \mu} + 1 \right).$$

Note that  $n_{\min} > 0$  by the choice of  $n_0, f$  and  $\mu$ .

(Proof of 1.) Assume  $\mathcal{D} \in \mathbf{P}$  and  $|D| = n > n_{\max}$ . Then there exists some  $i \in [m]$  and  $a_1^{\mathcal{D}}, \dots, a_{k_i}^{\mathcal{D}} \in \mathbb{N}$  such that  $h_r(\mathcal{D}) = \bar{v}_0^i + a_1^{\mathcal{D}} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}} \bar{v}_{k_i}^i$  (note that  $n = \|\bar{v}_0^i\|_1 + \sum_{j \in [k_i]} a_j^{\mathcal{D}} \|\bar{v}_j^i\|_1$ ). Let  $\mathcal{D}'$  be the  $\sigma$ -db with  $r$ -histogram  $\bar{v}_0^i + a_1^{\mathcal{D}'} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}'} \bar{v}_{k_i}^i \in M_i$  where  $a_j^{\mathcal{D}'}$  is the nearest integer to  $a_j^{\mathcal{D}} n_0/n$ , and hence  $a_j^{\mathcal{D}'} n_0/n - 1/2 \leq a_j^{\mathcal{D}} n_0/n \leq a_j^{\mathcal{D}'} n_0/n + 1/2$ . Note that since  $\bar{v}_0^i + a_1^{\mathcal{D}'} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}'} \bar{v}_{k_i}^i \in h_r(\mathbf{P})$ ,  $\mathcal{D}'$  exists and  $\mathcal{D}' \in \mathbf{P}$ . We need to show that  $n_{\min} \leq |D'| \leq n_{\max}$  and  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 \leq f - \mu$ .

▷ Claim 10 (\*).  $|D'| \geq n_{\min}$ .

▷ Claim 11 (\*).  $|D'| \leq n_{\max}$ .

▷ Claim 12.  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 \leq f - \mu$ .

**Proof.** By definition,  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 = \sum_{j \in [c]} |\text{dv}_r(\mathcal{D})[j] - \text{dv}_r(\mathcal{D}')[j]|$ . First recall that  $0 < n_0 - kv \leq |D'| \leq n_0 + kv < n$  and note that for every  $\ell \in [k_i]$ ,  $a_\ell^{\mathcal{D}} \leq n$  (since

**7:10 Faster Property Testers in a Variation of the Bounded Degree Model**

$\|\bar{v}_\ell^i\|_1 \neq 0$ ). Then for every  $j \in [c]$ , by the choice of  $a_\ell^{\mathcal{D}'}$  for  $\ell \in [k_i]$ ,

$$\begin{aligned} & \text{dv}_r(\mathcal{D})[j] - \text{dv}_r(\mathcal{D}') [j] = \bar{v}_0^i[j] \left( \frac{1}{n} - \frac{1}{|D'|} \right) + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}}}{n} - \frac{a_\ell^{\mathcal{D}'}}{|D'|} \right) \\ & \leq \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}}}{n} - \frac{a_\ell^{\mathcal{D}} n_0}{n|D'|} + \frac{1}{2|D'|} \right) = \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}}}{n} \left( \frac{|D'| - n_0}{|D'|} \right) + \frac{1}{2|D'|} \right) \\ & \leq \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{n}{n} \left( \frac{kv + n_0 - n_0}{n_0 - kv} \right) + \frac{1}{2(n_0 - kv)} \right) = \left( \frac{2kv + 1}{2(n_0 - kv)} \right) \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \\ & \leq \frac{kv(2kv + 1)}{n_0 - kv}. \end{aligned}$$

On the other hand,

$$\begin{aligned} & \text{dv}_r(\mathcal{D})[j] - \text{dv}_r(\mathcal{D}') [j] \geq -\frac{\bar{v}_0^i[j]}{|D'|} + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}}}{n} \left( \frac{|D'| - n_0}{|D'|} \right) - \frac{1}{2|D'|} \right) \\ & \geq -\frac{\bar{v}_0^i[j]}{|D'|} + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}}}{n} \left( \frac{-kv + n_0 - n_0}{|D'|} \right) - \frac{1}{2|D'|} \right) \\ & = -\frac{\bar{v}_0^i[j]}{|D'|} - \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}} kv}{n|D'|} + \frac{1}{2|D'|} \right) \\ & \geq -\frac{\bar{v}_0^i[j]}{n_0 - kv} - \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{nk v}{n(n_0 - kv)} + \frac{1}{2(n_0 - kv)} \right) \\ & = -\frac{\bar{v}_0^i[j]}{n_0 - kv} - \left( \frac{2kv + 1}{2(n_0 - kv)} \right) \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \geq -\frac{kv(2kv + 1)}{n_0 - kv}. \end{aligned}$$

Hence,

$$|\text{dv}_r(\mathcal{D})[j] - \text{dv}_r(\mathcal{D}') [j]| \leq \frac{kv(2kv + 1)}{n_0 - kv} \leq \frac{3(kv)^2}{n_0 - kv} = \frac{f - \mu}{c}$$

by the choice of  $n_0$ . Therefore,

$$\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 = \sum_{j \in [c]} |\text{dv}_r(\mathcal{D})[j] - \text{dv}_r(\mathcal{D}') [j]| \leq f - \mu$$

as required.  $\triangleleft$

(Proof of 2.) Assume  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$  and  $|D| = n > n_{\max}$ . For a contradiction let us assume there does exist a  $\sigma$ -db  $\mathcal{D}' \in \mathbf{P}$  such that  $n_{\min} \leq |D'| \leq n_{\max}$  and  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 \leq f + \mu$ . As  $\mathcal{D}' \in \mathbf{P}$  there exists some  $i \in [m]$  and  $a_1^{\mathcal{D}'}, \dots, a_{k_i}^{\mathcal{D}'} \in \mathbb{N}$  such that  $\text{h}_r(\mathcal{D}') = \bar{v}_0^i + a_1^{\mathcal{D}'} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}'} \bar{v}_{k_i}^i$ . Let  $\mathcal{D}''$  be the  $\sigma$ -db with  $r$ -histogram  $\bar{v}_0^i + a_1^{\mathcal{D}''} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}''} \bar{v}_{k_i}^i \in M_i$  where  $a_j^{\mathcal{D}''}$  is the nearest integer to  $a_j^{\mathcal{D}'} n / |D'|$ . Note as  $\bar{v}_0^i + a_1^{\mathcal{D}''} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}''} \bar{v}_{k_i}^i \in \text{h}_r(\mathbf{P})$ ,  $\mathcal{D}''$  exists and  $\mathcal{D}'' \in \mathbf{P}$ .

$\triangleright$  **Claim 13.**  $\mathcal{D}$  is  $\epsilon$ -close to  $\mathbf{P}$ .

*Proof.* First note that as  $\|\text{dv}_r(\mathcal{D}) - \text{dv}_r(\mathcal{D}')\|_1 \leq f + \mu$  and  $\text{h}_r(\mathcal{D}') = \bar{v}_0^i + a_1^{\mathcal{D}'} \bar{v}_1^i + \dots + a_{k_i}^{\mathcal{D}'} \bar{v}_{k_i}^i$ , for every  $j \in [c]$

$$\frac{\bar{v}_0^i[j] + \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}'} \bar{v}_\ell^i[j]}{|D'|} - f - \mu \leq \text{dv}_r(\mathcal{D})[j] \leq \frac{\bar{v}_0^i[j] + \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}'} \bar{v}_\ell^i[j]}{|D'|} + f + \mu$$

and therefore

$$n \left( \frac{\bar{v}_0^i[j] + \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}'} \bar{v}_\ell^i[j]}{|D'|} - f - \mu \right) \leq h_r(\mathcal{D})[j] \leq n \left( \frac{\bar{v}_0^i[j] + \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}'} \bar{v}_\ell^i[j]}{|D'|} + f + \mu \right).$$

Hence, by the choice of  $a_\ell^{\mathcal{D}''}$  for  $\ell \in [k_i]$ ,

$$\begin{aligned} h_r(\mathcal{D})[j] - h_r(\mathcal{D}'')[j] &\leq \bar{v}_0^i[j] \left( \frac{n}{|D'|} - 1 \right) + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - a_\ell^{\mathcal{D}''} \right) + fn + \mu n \\ &\leq \bar{v}_0^i[j] \frac{n}{|D'|} + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - \frac{1}{2} \right) \right) + fn + \mu n \\ &= \bar{v}_0^i[j] \frac{n}{|D'|} + \frac{1}{2} \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] + fn + \mu n. \end{aligned}$$

Similarly, by the choice of  $a_\ell^{\mathcal{D}''}$  for  $\ell \in [k_i]$  and as  $n > |D'|$ ,

$$\begin{aligned} h_r(\mathcal{D})[j] - h_r(\mathcal{D}'')[j] &\geq \bar{v}_0^i[j] \left( \frac{n}{|D'|} - 1 \right) + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - a_\ell^{\mathcal{D}''} \right) - fn - \mu n \\ &\geq -\bar{v}_0^i[j] \frac{n}{|D'|} + \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} + \frac{1}{2} \right) \right) - fn - \mu n \\ &= -\bar{v}_0^i[j] \frac{n}{|D'|} - \frac{1}{2} \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] - fn - \mu n. \end{aligned}$$

Therefore,

$$\begin{aligned} |h_r(\mathcal{D})[j] - h_r(\mathcal{D}'')[j]| &\leq \bar{v}_0^i[j] \frac{n}{|D'|} + \frac{1}{2} \sum_{\ell \in [k_i]} \bar{v}_\ell^i[j] + fn + \mu n \\ &\leq \frac{n}{|D'|} \sum_{0 \leq \ell \leq k_i} \bar{v}_\ell^i[j] + fn + \mu n \leq \frac{nk v}{|D'|} + fn + \mu n \\ &= n \left( \frac{k v}{|D'|} + \frac{\lambda}{3c} + \frac{\lambda}{6c} \right) \leq n \left( \frac{\lambda}{18c} + \frac{\lambda}{3c} + \frac{\lambda}{6c} \right) = \frac{5\lambda n}{9c} \end{aligned}$$

by the choice of  $f$  and  $\mu$  and as

$$|D'| \geq n_{\min} = \frac{3c(kv)^2}{f - \mu} = \frac{18(ckv)^2}{\lambda} \geq \frac{18ckv}{\lambda}.$$

To apply Lemma 8 we need to show that  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}'')\|_1 \leq \lambda \min\{n, |D''|\}$ . If  $|h_r(\mathcal{D})[j] - h_r(\mathcal{D}'')[j]| \leq \frac{\lambda}{c} \min\{n, |D''|\}$  then  $\|h_r(\mathcal{D}) - h_r(\mathcal{D}'')\|_1 \leq \lambda \min\{n, |D''|\}$ . Clearly,  $\frac{5\lambda n}{9c} < \frac{\lambda n}{c}$ . We also have

$$\begin{aligned} |D''| &= \|\bar{v}_0^i\|_1 + \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}''} \|\bar{v}_\ell^i\|_1 \geq \|\bar{v}_0^i\|_1 + \sum_{\ell \in [k_i]} \left( \frac{a_\ell^{\mathcal{D}'} n}{|D'|} - \frac{1}{2} \right) \|\bar{v}_\ell^i\|_1 \\ &= \|\bar{v}_0^i\|_1 - \frac{1}{2} \sum_{\ell \in [k_i]} \|\bar{v}_\ell^i\|_1 + \frac{n}{|D'|} \sum_{\ell \in [k_i]} a_\ell^{\mathcal{D}'} \|\bar{v}_\ell^i\|_1 \geq -kv + \frac{n}{|D'|} (|D'| - \|\bar{v}_0^i\|_1) \\ &\geq -\frac{n}{18} + \frac{17}{18}n \geq \frac{5n}{9} \end{aligned}$$

## 7:12 Faster Property Testers in a Variation of the Bounded Degree Model

as

$$|D'| \geq \frac{18ckv}{\lambda} \geq 18v \geq 18\|\bar{v}_0^i\|_1 \text{ and } kv \leq \frac{(ckv)^2}{\lambda} = \frac{n_{\min}}{18} \leq \frac{n}{18}.$$

Therefore,  $\frac{5\lambda n}{9c} \leq \frac{\lambda|D'|}{c}$  and hence  $\|\mathbf{h}_r(\mathcal{D}) - \mathbf{h}_r(\mathcal{D}')\|_1 \leq \lambda \min\{n, |D'|\}$ . By Lemma 8,  $\mathcal{D}$  is  $\epsilon$ -close to  $\mathbf{P}$ .  $\triangleleft$

Claim 13 gives us a contradiction and therefore for every  $\mathcal{D}' \in \mathbf{P}$  such that  $n_{\min} \leq |D'| \leq n_{\max}$ , we have  $\|\mathbf{d}_{v_r}(\mathcal{D}) - \mathbf{d}_{v_r}(\mathcal{D}')\|_1 > f + \mu$  as required.  $\blacktriangleleft$

As mentioned in the introduction, Alon [22, Proposition 19.10] proved that on bounded degree graphs, for any graph  $\mathcal{G}$ , radius  $r$  and  $\epsilon > 0$  there always exists a graph  $\mathcal{H}$  whose size is independent of  $|V(\mathcal{G})|$  and whose  $r$ -neighbourhood distribution vector satisfies  $\|\mathbf{d}_{v_r}(\mathcal{G}) - \mathbf{d}_{v_r}(\mathcal{H})\|_1 \leq \epsilon$ . However, the proof is only existential and does not provide an explicit bound on the size of  $\mathcal{H}$ . As a corollary to the proof of Theorem 9, we immediately obtain explicit bounds for classes of graphs and relational databases of bounded degree whose histogram vectors form a semilinear set.

► **Corollary 14.** *Let  $\epsilon \in (0, 1]$ ,  $r \in \mathbb{N}$  and  $\mathcal{D}$  be a  $\sigma$ -db that belongs to a class of  $\sigma$ -dbs  $\mathbf{C}$  such that the set  $\mathbf{h}_r(\mathbf{C})$  is semilinear, i.e.  $\mathbf{h}_r(\mathbf{C}) = M_1 \cup M_2 \cup \dots \cup M_m$  where  $m \in \mathbb{N}$  and for each  $i \in [m]$ ,  $M_i = \{\bar{v}_0^i + a_1\bar{v}_1^i + \dots + a_{k_i}\bar{v}_{k_i}^i \mid a_1, \dots, a_{k_i} \in \mathbb{N}\}$  is a linear set where  $\bar{v}_0^i, \dots, \bar{v}_{k_i}^i \in \mathbb{N}^{c(r)}$ . Then there exists a  $\sigma$ -db  $\mathcal{D}_0$  such that*

$$\|\mathbf{d}_{v_r}(\mathcal{D}) - \mathbf{d}_{v_r}(\mathcal{D}_0)\|_1 \leq \epsilon \text{ and } |D_0| \leq kv \left( \frac{3ckv}{\epsilon} + 2 \right)$$

where  $c := c(r)$ ,  $k := \max_{i \in [m]} k_i + 1$  and  $v := \max_{i \in [m]} \left( \max_{j \in [0, k_i]} \|\bar{v}_j^i\|_1 \right)$ .

Our aim is to construct constant time testers for hyperfinite properties whose set of  $r$ -histograms are semilinear. If we can approximate the  $r$ -neighbourhood distribution of a  $\sigma$ -db then by Theorem 9 we only need to check whether this distribution is close or not to the  $r$ -neighbourhood distribution of some small constant size  $\sigma$ -db. We let  $\text{EstimateFrequencies}_{r,s}$  be the algorithm that, given oracle access to an input  $\sigma$ -db  $\mathcal{D}$ , samples  $s$  many elements uniformly and independently from  $D$  and computes their  $r$ -type. The algorithm then returns the  $r$ -neighbourhood distribution vector of the sample.

► **Lemma 15** ([1]). *Let  $\mathcal{D} \in \mathbf{C}_d$  be a  $\sigma$ -db on  $n$  elements,  $\mu \in (0, 1)$  and  $r \in \mathbb{N}$ . If  $s \geq c(r)^2/\mu^2 \cdot \ln(20c(r))$ , with probability at least  $9/10$  the vector  $\bar{v}$  returned by the algorithm  $\text{EstimateFrequencies}_{r,s}$  on input  $\mathcal{D}$  satisfies  $\|\bar{v} - \mathbf{d}_{v_r}(\mathcal{D})\|_1 \leq \mu$ .*

► **Theorem 16.** *Let  $\mathbf{C}$  be closed under removing tuples and let  $\mathbf{P} \subseteq \mathbf{C}$  be a property that is hyperfinite on  $\mathbf{C}$ . If for each  $r \in \mathbb{N}$  the set  $\mathbf{h}_r(\mathbf{P})$  is semilinear, then  $\mathbf{P}$  is uniformly testable on  $\mathbf{C}$  in constant time in the  $\text{BDRD}_{+/-}$  model.*

**Proof.** Let  $\epsilon \in (0, 1]$ . Let  $r := r(\epsilon)$  be as in Lemma 8, let  $n_{\min} := n_{\min}(\epsilon)$ ,  $n_{\max} := n_{\max}(\epsilon)$ ,  $f := f(\epsilon)$  and  $\mu := \mu(\epsilon)$  be as in Theorem 9 and let  $s = c(r)^2/\mu^2 \cdot \ln(20c(r))$ . Assume that the set  $\mathbf{h}_r(\mathbf{P})$  is semilinear. Given oracle access to a  $\sigma$ -db  $\mathcal{D} \in \mathbf{C}$  and  $|D| = n$  as an input, the  $\epsilon$ -tester proceeds as follows:

1. If  $n \leq n_{\max}$ , do a full check of  $\mathcal{D}$  and decide if  $\mathcal{D} \in \mathbf{P}$ .
2. Run  $\text{EstimateFrequencies}_{r,s}$  and let  $\bar{v}$  be the resulting vector.
3. If there exists a  $\mathcal{D}' \in \mathbf{P}$  where  $n_{\min} \leq |D'| \leq n_{\max}$  and  $\|\bar{v} - \mathbf{d}_{v_r}(\mathcal{D}')\|_1 \leq f$  then accept otherwise reject.

The running time and query complexity of the above tester is constant as  $n_{\max}$  is a constant (it only depends on  $\mathbf{P}$ ,  $d$  and  $\epsilon$ ) and  $\text{EstimateFrequencies}_{r,s}$  runs in constant time and makes a constant number of queries.

For correctness, first assume  $\mathcal{D} \in \mathbf{P}$ . By Theorem 9 there exists a  $\sigma$ -db  $\mathcal{D}' \in \mathbf{P}$  such that  $n_{\min} \leq |D'| \leq n_{\max}$  and  $\|dv_r(\mathcal{D}) - dv_r(\mathcal{D}')\|_1 \leq f - \mu$ . By Lemma 15 with probability at least  $9/10$ ,  $\|\bar{v} - dv_r(\mathcal{D})\|_1 \leq \mu$  and therefore  $\|\bar{v} - dv_r(\mathcal{D}')\|_1 \leq f$ . Hence with probability at least  $9/10$  the tester will accept.

Now assume  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$ . By Theorem 9 for every  $\mathcal{D}' \in \mathbf{P}$  with  $n_{\min} \leq |D'| \leq n_{\max}$ , we have  $\|dv_r(\mathcal{D}) - dv_r(\mathcal{D}')\|_1 > f + \mu$ . By Lemma 15 with probability at least  $9/10$ ,  $\|\bar{v} - dv_r(\mathcal{D})\|_1 \leq \mu$  and therefore for every  $\mathcal{D}' \in \mathbf{P}$  with  $n_{\min} \leq |D'| \leq n_{\max}$ ,  $\|\bar{v} - dv_r(\mathcal{D}')\|_1 > f$ . Hence with probability at least  $9/10$  the tester will reject. ◀

Combining Theorem 16 and Lemma 1 and the fact that  $\mathbf{C}_d^t$  is hyperfinite [17, 3] (and so any property is hyperfinite on  $\mathbf{C}_d^t$ ) we obtain the following as a corollary.

► **Theorem 17.** *Every property  $\mathbf{P}$  definable by a CMSO sentence on  $\mathbf{C}_d^t$  is uniformly testable on  $\mathbf{C}_d^t$  with constant time complexity in the  $\text{BDRD}_{+/-}$  model.*

---

## References

- 1 Isolde Adler and Frederik Harwath. Property testing for bounded degree databases. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96, page 6. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.
- 2 Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008.
- 3 Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 293–299. ACM, 1990. doi:10.1145/100216.100254.
- 4 Jasine Babu, Areej Khoury, and Ilan Newman. Every property of outerplanar graphs is testable. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 5 Sagi Ben-Moshe, Yaron Kanza, Eldar Fischer, Arie Matsliah, Mani Fischer, and Carl Staelin. Detecting and exploiting near-sortedness for efficient relational query evaluation. In *Proceedings of the 14th International Conference on Database Theory*, pages 256–267. ACM, 2011.
- 6 Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. *Advances in mathematics*, 223(6):2200–2218, 2010.
- 7 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering fo+ mod queries under updates on bounded degree databases. *ACM Transactions on Database Systems (TODS)*, 43(2):7, 2018.
- 8 Hubie Chen and Yuichi Yoshida. Testability of homomorphism inadmissibility: Property testing meets database theory. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 365–382. ACM, 2019.
- 9 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, pages 193–242. Elsevier, 1990.
- 10 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.
- 11 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic (TOCL)*, 8(4):21, 2007.

- 12 Hendrik Fichtenberger, Pan Peng, and Christian Sohler. On constant-size graphs that preserve the local structure of high-girth graphs. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPICs*, pages 786–799. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.APPROX-RANDOM.2015.786.
- 13 Hendrik Fichtenberger, Pan Peng, and Christian Sohler. Every testable (infinite) property of bounded-degree graphs contains an infinite hyperfinite subproperty. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 714–726. SIAM, 2019.
- 14 Eldar Fischer, Johann A Makowsky, et al. On spectra of sentences of monadic second order logic with counting. *Journal of Symbolic Logic*, 69(3):617–640, 2004.
- 15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, Berlin, Heidelberg, 2006.
- 16 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.
- 17 Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 22–31. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.77.
- 18 Piotr Indyk, Andrew McGregor, Ilan Newman, and Krzysztof Onak. Open problems in data streams, property testing, and related topics. In *Bernitoto Workshop on Sublinear Algorithms*, 2011.
- 19 Hiro Ito, Areej Khoury, and Ilan Newman. On the characterization of 1-sided error strongly testable graph properties for bounded-degree graphs. *computational complexity*, 29(1):1, 2020.
- 20 Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011. doi:10.2168/LMCS-7(2:20)2011.
- 21 Leonid Libkin. *Elements of finite model theory*. Springer, 2004.
- 22 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.
- 23 Ilan Newman and Christian Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013.

## **A** Proofs of Section 3

**Proof of Lemma 4.** Let  $\pi$  be an  $\epsilon$ -tester, that runs in time  $f(n)$ , for  $\mathbf{P}$  on  $\mathbf{C}$  in the BDRD model. We claim that  $\pi$  is also an  $\epsilon$ -tester for  $\mathbf{P}$  on  $\mathbf{C}$  in the  $\text{BDRD}_{+/-}$  model. Let  $\mathcal{D} \in \mathbf{C}$  be the input  $\sigma$ -db. If  $\mathcal{D} \in \mathbf{P}$  then  $\pi$  will accept with probability at least  $2/3$ . If  $\mathcal{D}$  is  $\epsilon$ -far from  $\mathbf{P}$  in the  $\text{BDRD}_{+/-}$  model then it must also be  $\epsilon$ -far from  $\mathbf{P}$  in the BDRD model and therefore  $\pi$  will reject with probability at least  $2/3$ . Hence  $\pi$  is an  $\epsilon$ -tester for  $\mathbf{P}$  on  $\mathbf{C}$  in the  $\text{BDRD}_{+/-}$  model. ◀

**B Proofs of Section 4**

Proof of Claim 10. By the choice of  $a_j^{\mathcal{D}'}$  for  $j \in [k_i]$ ,

$$\begin{aligned} |D'| &= \|\bar{v}_0^i\|_1 + \sum_{j \in [k_i]} a_j^{\mathcal{D}'} \|\bar{v}_j^i\|_1 \geq \|\bar{v}_0^i\|_1 + \sum_{j \in [k_i]} \left( \frac{a_j^{\mathcal{D}} n_0}{n} - \frac{1}{2} \right) \|\bar{v}_j^i\|_1 \\ &= \|\bar{v}_0^i\|_1 - \frac{1}{2} \sum_{j \in [k_i]} \|\bar{v}_j^i\|_1 + \frac{n_0}{n} \sum_{j \in [k_i]} a_j^{\mathcal{D}} \|\bar{v}_j^i\|_1 = \|\bar{v}_0^i\|_1 - \frac{1}{2} \sum_{j \in [k_i]} \|\bar{v}_j^i\|_1 + n_0 - \frac{n_0 \|\bar{v}_0^i\|_1}{n} \\ &\geq \|\bar{v}_0^i\|_1 - \frac{1}{2} \sum_{j \in [k_i]} \|\bar{v}_j^i\|_1 + n_0 - \|\bar{v}_0^i\|_1 \geq -kv + n_0 = n_{\min}, \end{aligned}$$

as  $\sum_{j \in [k_i]} a_j^{\mathcal{D}} \|\bar{v}_j^i\|_1 = n - \|\bar{v}_0^i\|_1$  and  $n > n_{\max} \geq n_0$ . ◁

Proof of Claim 11. By the choice of  $a_j^{\mathcal{D}'}$  for  $j \in [k_i]$ ,

$$\begin{aligned} |D'| &= \|\bar{v}_0^i\|_1 + \sum_{j \in [k_i]} a_j^{\mathcal{D}'} \|\bar{v}_j^i\|_1 \leq \|\bar{v}_0^i\|_1 + \sum_{j \in [k_i]} \left( \frac{a_j^{\mathcal{D}} n_0}{n} + \frac{1}{2} \right) \|\bar{v}_j^i\|_1 \\ &= \|\bar{v}_0^i\|_1 + \frac{1}{2} \sum_{j \in [k_i]} \|\bar{v}_j^i\|_1 + n_0 \left( 1 - \frac{\|\bar{v}_0^i\|_1}{n} \right) \leq \sum_{0 \leq j \leq k_i} \|\bar{v}_j^i\|_1 + n_0 \leq kv + n_0 = n_{\max}, \end{aligned}$$

as  $\sum_{j \in [k_i]} a_j^{\mathcal{D}} \|\bar{v}_j^i\|_1 = n - \|\bar{v}_0^i\|_1$ . ◁



# Clustering Under Perturbation Stability in Near-Linear Time

**Pankaj K. Agarwal**

Department of Computer Science, Duke University, Durham, NC, USA  
pankaj@cs.duke.edu

**Hsien-Chih Chang**

Department of Computer Science, Dartmouth College, Hanover, NH, USA  
hsien-chih.chang@dartmouth.edu

**Kamesh Munagala**

Department of Computer Science, Duke University, Durham, NC, USA  
kamesh@cs.duke.edu

**Erin Taylor**

Department of Computer Science, Duke University, Durham, NC, USA  
ect15@cs.duke.edu

**Emo Welzl**

Department of Computer Science, ETH Zürich, Switzerland  
emo@inf.ethz.ch

---

## Abstract

We consider the problem of center-based clustering in low-dimensional Euclidean spaces under the perturbation stability assumption. An instance is  $\alpha$ -stable if the underlying optimal clustering continues to remain optimal even when all pairwise distances are arbitrarily perturbed by a factor of at most  $\alpha$ . Our main contribution is in presenting efficient exact algorithms for  $\alpha$ -stable clustering instances whose running times depend near-linearly on the size of the data set when  $\alpha \geq 2 + \sqrt{3}$ . For  $k$ -center and  $k$ -means problems, our algorithms also achieve polynomial dependence on the number of clusters,  $k$ , when  $\alpha \geq 2 + \sqrt{3} + \varepsilon$  for any constant  $\varepsilon > 0$  in any fixed dimension. For  $k$ -median, our algorithms have polynomial dependence on  $k$  for  $\alpha > 5$  in any fixed dimension; and for  $\alpha \geq 2 + \sqrt{3}$  in two dimensions. Our algorithms are simple, and only require applying techniques such as local search or dynamic programming to a suitably modified metric space, combined with careful choice of data structures.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Design and analysis of algorithms

**Keywords and phrases** clustering, stability, local search, dynamic programming, coresets, polyhedral metric, trapezoid decomposition, range query

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.8

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2009.14358>.

**Funding** Work on this paper was partially supported by NSF grants IIS-18-14493, CCF-20-07556, CCF-1637397 and IIS-1447554; ONR award N00014-19-1-2268; and DARPA award FA8650-18-C-7880.

*Hsien-Chih Chang:* This work was done when the author was affiliated with Duke University.

## 1 Introduction

Clustering is a fundamental problem in unsupervised learning and data summarization, with wide-ranging applications that span myriad areas. Typically, the data points are assumed to lie in a Euclidean space, and the goal in center-based clustering is to open a set of  $k$  centers to minimize the objective cost, usually a function over the distance from each data point



© Pankaj K. Agarwal, Hsien-Chih Chang, Kamesh Munagala, Erin Taylor, and Emo Welzl; licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 8; pp. 8:1–8:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to its closest center. The  $k$ -median objective minimizes the sum of distances; the  $k$ -means minimizes the sum of squares of distances; and the  $k$ -center minimizes the longest distance. In the worst case, all these objectives are NP-hard even in 2D [48, 50].

A substantial body of work has focused on developing polynomial-time approximation algorithms and analyzing natural heuristics for these problems. Given the sheer size of modern data sets, such as those generated in genomics or mapping applications, even a polynomial-time algorithm is too slow to be useful in practice – just computing all pairs of distances can be computationally burdensome. What we need is an algorithm whose running time is near-linear in the input size and polynomial in the number of clusters.

Because of NP-hardness results, we cannot hope to compute an optimal solution in polynomial time, but in the worst case an approximate clustering can be different from an optimal clustering. We focus on the case when the optimal clustering can be recovered under some reasonable assumptions on the input that hold in practice. Such methodology is termed “beyond worst-case analysis” and has been adopted by recent work [2, 8, 23]. In recent years, the notion of *stability* has emerged as a popular assumption under which polynomial-time optimal clustering algorithms have been developed. An instance of clustering is called *stable* if any “small perturbation” of input points does not change the optimal solution. This is natural in real datasets, where often, the optimal clustering is clearly demarcated, and the distances are obtained heuristically. Different notions of stability differ in how “small perturbation” is defined, though most of them are related. In this paper, we focus on the notions of stability introduced in Bilu and Linial [23] and Awasthi, Blum, and Sheffet [14]. A clustering instance is  $\alpha$ -*perturbation resilient* or  $\alpha$ -*stable* if the optimal clustering does not change when all distances are perturbed by a factor of at most  $\alpha$ . Similarly, a clustering instance is  $\alpha$ -*center proximal* if any point is at least a factor of  $\alpha$  closer to its own optimal center than any other optimal center. Awasthi, Blum, and Sheffet showed that  $\alpha$ -stability implies  $\alpha$ -center proximity [14]. This line of work designs algorithms to recover the *exact* optimal clustering – the ground truth – in polynomial time for  $\alpha$ -stable instances.

This paper also focuses on recovering the optimal clustering for stable clustering instances. But instead of focusing on polynomial-time algorithms and optimizing the value of  $\alpha$ , we ask the question: *Can algorithms be designed that compute exact solutions to stable instances of Euclidean center-based clustering that run in time near-linear in the input size?* We note that an  $(1 + \epsilon)$ -approximation solution, for an arbitrarily small constant  $\epsilon > 0$ , may differ significantly from an optimal solution (the ground truth) even for stable instances, so one cannot hope to use an approximation algorithm to recover the optimal clustering.

## 1.1 Our Results

In this paper, we make progress on the above question, and present near-linear time algorithms for finding optimal solutions of stable clustering instances with moderate values of  $\alpha$ . In particular, we show the following meta-theorem:

► **Theorem 1.** *Let  $X$  be a set of  $n$  points in  $\mathbb{R}^d$  for some constant  $d$ , let  $k \geq 1$  be an integer, and let  $\alpha \geq 2 + \sqrt{3}$  be a parameter. If the  $k$ -median,  $k$ -means, or  $k$ -center clustering instance for  $X$  is  $\alpha$ -stable, then the optimal solution can be computed in  $\tilde{O}(n \text{ poly } k + f(k))$  time.*

In the above theorem, the  $\tilde{O}$  notation suppresses logarithmic terms in  $n$  and the spread of the point set. The function  $f(k)$  depends on the choice of algorithm, and we present the exact dependence below. We also omit terms depending solely on the dimension,  $d$ . Furthermore, the above theorem is robust in the sense that the algorithm is not restricted to choosing the input points as centers (*discrete setting*), and can potentially choose arbitrary

points in the Euclidean plane as centers (*continuous setting*, sometimes referred to as the *Steiner point setting*) – indeed, we show that these notions are identical under a reasonable assumption on stability.

At a more fine-grained level, we present several algorithms that require mild assumptions on the stability condition. In the results below, as well as throughout the paper, we present our results both for the Euclidean plane, as well as generalizations to higher (but fixed number of) dimensions.

**Dynamic Programming.** In Section 3, we present a dynamic programming algorithm that computes the optimal clustering in  $O(nk^2 + n \text{polylog } n)$  time for  $\alpha$ -stable  $k$ -means,  $k$ -median, and  $k$ -center in any fixed dimension, provided that  $\alpha \geq 2 + \sqrt{3} + \varepsilon$  for any constant  $\varepsilon > 0$ . For  $d = 2$ , it suffices to assume that  $\alpha \geq 2 + \sqrt{3}$ .

**Local Search.** In Sections 4 and 5, we show that the standard 1-swap local-search algorithm, which iteratively swaps out a center in the current solution for a new center as long as the resulting total cost improves, computes an optimal clustering for  $\alpha$ -stable instances of  $k$ -median assuming  $\alpha > 5$ . We also show that it can be implemented in  $O(nk^2 \log^3 n \log \Delta)$  for  $d = 2$  and in  $O(nk^{2d-1} \text{polylog } n \log \Delta)$  for  $d > 2$ ;  $\Delta$  is the spread of the point set.<sup>1</sup>

**Coresets.** In the full version of the paper, we use multiplicative coresets to compute the optimal clustering for  $k$ -means,  $k$ -median and  $k$ -center in any fixed dimension, when  $\alpha \geq 2 + \sqrt{3}$ . The running time is  $O(nk^2 + f(k))$  where  $f(k)$  is an exponential function of  $k$ .

► **Remark 2.** While the current analysis of the dynamic programming based algorithm suggests that it is better than the local-search and coreset based approaches, the latter are of independent interest – our local-search analysis is considerably simpler than the previous analysis [38], and coresets have mostly been used to compute approximate, rather than exact, solutions. We also note that our analysis of the local-search algorithm is probably not tight. Furthermore, variants of all three approaches might work for smaller values of  $\alpha$ .

**Techniques.** The key difficulty with developing fast algorithms for computing the optimal clustering is that some clusters could have a very small size compared to others. This issue persists even when the instances are stable. Imagine a scenario where there are multiple small clusters, and an algorithm must decide whether to merge these into one cluster while splitting some large cluster, or keep them intact. Now imagine this situation happening recursively, so that the algorithm has multiple choices about which clusters to recursively split. The difference in cost between these options and the size of the small clusters can be small enough that any  $(1 + \epsilon)$ -approximation can be agnostic, while an exact solution cannot. As such, work on finding exact optima use techniques such as dynamic programming [10] or local search with large number of swaps [26, 38] in order to recover small clusters. Other work makes assumptions lower-bounding the size of the optimal clusters or the spread of their centers [34].

Our main technical insight for the first two results is simple in hindsight, yet powerful: For a stable instance, if the Euclidean metric is replaced by another metric that is a good approximation, then the optimal clustering does not change under the new metric and in fact the instance remains stable albeit with a smaller stability parameter. In particular,

---

<sup>1</sup> The *spread* of a point set is the ratio between the longest and shortest pairwise distances.

we replace the Euclidean metric with an appropriate *polyhedral metric* – that is, a convex distance function where each unit ball is a regular polyhedron – yielding efficient procedures for the following two primitives:

- **Cost of 1-swap.** Given a candidate set of centers  $S$ , maintain a data structure that efficiently updates the total cost if center  $x \in S$  is replaced by center  $y \notin S$ .
- **Cost of 1-clustering.** Given a partition of the data points, maintain a data structure where the cost of 1-clustering (under any objectives) can be efficiently updated as partitions are merged.

We next combine the insight of changing the metrics with additional techniques. For local search, we build on the approach in [26,31,38] that shows local search with  $t$ -swaps for large enough constant  $t$  finds an optimal solution for stable instances in polynomial time for any fixed-dimension Euclidean space. None of the prior analysis directly extends as is to 1-swap, which is critical in achieving near-linear running time – note that even when  $t = 2$  there is a quadratic number of candidate swaps per step.

For the dynamic programming algorithm, we use the following insight: In Euclidean spaces, for  $\alpha \geq 2 + \sqrt{3}$ , the longest edge of the minimum spanning tree over the input points partitions the data set in two, such that any optimal cluster lies completely in one of the two sides of the partition. Combined with the change of metrics one can achieve near-linear running time.

Due to length constraints of the paper, the coresets result, most of algorithmic details, and many proofs can be found in the full version of the paper.

## 1.2 Related Work

All of  $k$ -median,  $k$ -means, and  $k$ -center are widely studied from the perspective of approximation algorithms and are known to be hard to approximate [36]. Indeed, for general metric spaces,  $k$ -center is hard to approximate to within a factor of  $2 - \epsilon$  [43];  $k$ -median is hard to  $(1 + 2/e)$ -approximate [44]; and  $k$ -means is hard to  $(1 + 8/e)$ -approximate in general metrics [29], and is hard to approximate within a factor of 1.0013 in the Euclidean setting [47]. Even when the metric space is Euclidean,  $k$ -means is still NP-hard when  $k = 2$  [7,32], and there is an  $n^{\Omega(k)}$  lower bound on running time for  $k$ -median and  $k$ -means in 4-dimensional Euclidean space under the exponential-time hypothesis [27].

There is a long line of work in developing  $(1 + \epsilon)$ -approximations for these problems in Euclidean spaces. The holy grail of this work has been the development of algorithms that are near-linear time in  $n$ , and several techniques are now known to achieve this. This includes randomly shifted quad-trees [11], coresets [4,15,37,40,41], sampling [46], and local search [26,28,30], among others.

There are many notions of clustering stability that have been considered in literature [1,6,13,17,18,22,35,45,52]. The exact definition of stability we study here was first introduced in Awasthi et al. [14]; their definition in particular resembles the one of Bilu and Linial [23] for max-cut problem, which later has been adapted to other optimization problems [9,10,19,49,51]. Building on a long line of work [14,16,20,21], which gradually reduced the stability parameter, Angelidakis et al. [10] present a dynamic programming based polynomial-time optimal algorithm for discrete 2-stable instances for all center-based objectives.

Chekuri and Gupta [25] show that a natural LP-relaxation is integral for the 2-stable  $k$ -center problem. Recent work by Cohen-Addad [31] provides a framework for analyzing local search algorithms for stable instances. This work shows that for an  $\alpha$ -stable instance with  $\alpha > 3$ , any solution is optimal if it cannot be improved by swapping  $\lceil 2/(\alpha - 3) \rceil$

centers. Focusing on Euclidean spaces of fixed dimensions, Friggstad et al. [38] show that a local-search algorithm with  $O(1)$ -swaps runs in polynomial time under a  $(1 + \delta)$ -stable assumption for any  $\delta > 0$ . However, none of the algorithms for stable instances of clustering so far have running time near-linear in  $n$ , even when the stability parameter  $\alpha$  is large, points lie in  $\mathbb{R}^2$ , and the underlying metric is Euclidean.

On the hardness side, solving  $(3 - \delta)$ -center proximal  $k$ -median instances in general metric spaces is NP-hard for any  $\delta > 0$  [14]. When restricted to Euclidean spaces in arbitrary dimensions, Ben-David and Reyzin [22] showed that for every  $\delta > 0$ , solving discrete  $(2 - \delta)$ -center proximal  $k$ -median instances is NP-hard. Similarly, the clustering problem for discrete  $k$ -center remains hard for  $\alpha$ -stable instances when  $\alpha < 2$ , assuming standard complexity assumption that  $\text{NP} \neq \text{RP}$  [20]. Under the same complexity assumption, discrete  $\alpha$ -stable  $k$ -means is also hard when  $\alpha < 1 + \delta_0$  for some positive constant  $\delta_0$  [38]. Deshpande et al. [34] showed it is NP-hard to  $(1 + \epsilon)$ -approximate  $(2 - \delta)$ -center proximal  $k$ -means instances.

## 2 Definitions and Preliminaries

**Clustering.** Let  $X = \{p_1, \dots, p_n\}$  be a set of  $n$  points in  $\mathbb{R}^d$ , and let  $\delta: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$  be a distance function (not necessarily a metric satisfying triangle inequality). For a set  $Y \subseteq \mathbb{R}^d$ , we define  $\delta(p, Y) := \min_{y \in Y} \delta(p, y)$ . A  *$k$ -clustering* of  $X$  is a partition of  $X$  into  $k$  non-empty *clusters*  $X_1, \dots, X_k$ . We focus on center-based clusterings that are induced by a set  $S := \{c_1, \dots, c_k\}$  of  $k$  *centers*; each  $X_i$  is the subset of points of  $X$  that are closest to  $c_i$  in  $S$  under  $\delta$ , that is,  $X_i := \{p \in X \mid \delta(p, c_i) \leq \delta(p, c_j)\}$  (ties are broken arbitrarily). Assuming the nearest neighbor of each point of  $X$  in  $S$  is unique (under distance function  $\delta$ ),  $S$  defines a  $k$ -clustering of  $X$ . Sometimes it is more convenient to denote a  $k$ -clustering by its set of centers  $S$ .

The quality of a clustering  $S$  of  $X$  is defined using a cost function  $\$(X, S)$ ; cost function  $\$$  depends on the distance function  $\delta$ , so sometimes we may use the notation  $\$_\delta$  to emphasize the underlying distance function. The goal is to compute  $S^* := \arg \min_S \$(X, S)$  where the minimum is taken over all subsets  $S \subset \mathbb{R}^d$  of  $k$  points. Several different cost functions have been proposed, leading to various optimization problems. We consider the following three popular variants:

- *$k$ -median clustering*: the cost function is  $\$(X, S) = \sum_{p \in X} \delta(p, S)$ .
- *$k$ -means clustering*: the cost function is  $\$(X, S) = \sum_{p \in X} (\delta(p, S))^2$ .
- *$k$ -center clustering*: the cost function is  $\$(X, S) = \max_{p \in X} \delta(p, S)$ .

In some cases we wish  $S$  to be a subset of  $X$ , in which case we refer to the problem as the *discrete  $k$ -clustering* problem. For example, the discrete  $k$ -median problem is to compute  $\arg \min_{S \subseteq X, |S|=k} \sum_{p \in X} \delta(p, S)$ . The discrete  $k$ -means and discrete  $k$ -center problems are defined analogously.

Given point set  $X$ , distance function  $\delta$ , and cost function  $\$$ , we refer to  $(X, \delta, \$)$  as a *clustering instance*. If  $\$$  is defined directly by the distance function  $\delta$ , we use  $(X, \delta)$  to denote a clustering instance. Note that a center of a set of points may not be unique (e.g. when  $\delta$  is defined by the  $L_1$ -metric and  $\$$  is the sum of distances) or it may not be easy to compute (e.g. when  $\delta$  is defined by the  $L_2$ -metric and  $\$$  is the sum of distances).

**Stability.** Let  $X$  be a point set in Euclidean space  $\mathbb{R}^d$ . For  $\alpha \geq 1$ , a clustering instance  $(X, \delta, \$_\delta)$  is  *$\alpha$ -stable* if for any *perturbed distance function*  $\tilde{\delta}$  (not necessary a metric) satisfying  $\delta(p, q) \leq \tilde{\delta}(p, q) \leq \alpha \cdot \delta(p, q)$  for all  $p, q \in \mathbb{R}^d$ , any optimal clustering of  $(X, \delta, \$_\delta)$  is also an optimal clustering of  $(X, \tilde{\delta}, \$_{\tilde{\delta}})$ . Note that the cluster centers as well as the cost of optimal clustering may be different for the two instances. We exploit the following property of stability, which follows directly from its definition.

## 8:6 Clustering Under Perturbation Stability in Near-Linear Time

► **Lemma 3.** *Let  $(X, \delta)$  be an  $\alpha$ -stable clustering instance with  $\alpha > 1$ . Then the optimal clustering  $O$  of  $(X, \delta)$  is unique.*

**Metric approximations.** The next lemma, which we rely on heavily throughout the paper, is the observation that a change of metric preserves the optimal clustering as long as the new metric is a  $\beta$ -approximation of the original metric satisfying  $\beta < \alpha$ .

► **Lemma 4.** *Given point set  $X$ , let  $\delta$  and  $\delta'$  be two metrics satisfying  $\delta(p, q) \leq \delta'(p, q) \leq \beta \cdot \delta(p, q)$  for all  $p$  and  $q$  in  $X$  for some  $\beta$ . Let  $(X, \delta)$  be an  $\alpha$ -stable clustering instance with  $\alpha > \beta$ . Then the optimal clustering of  $(X, \delta)$  is also the optimal clustering of  $(X, \delta')$ , and vice versa. Furthermore,  $(X, \delta')$  is an  $(\alpha/\beta)$ -stable clustering instance.*

**Polyhedral metric.** In light of the metric approximation lemma, we would like to approximate the Euclidean metric without losing too much stability, using a collection of convex distance functions generalizing the  $L_\infty$ -metric in Euclidean space. Let  $N \subseteq S^{d-1}$  be a centrally-symmetric set of  $\gamma$  unit vectors (that is, if  $u \in N$  then  $-u \in N$ ) such that for any unit vector  $v \in S^{d-1}$ , there is a vector  $u \in N$  within angle  $\arccos(1 - \epsilon) = O(\sqrt{\epsilon})$ . The number of vectors needed in  $N$  is known to be  $O(\epsilon^{-(d-1)/2})$ . We define the *polyhedral metric*  $\delta_N: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$  to be  $\delta_N(p, q) := \max_{u \in N} \langle p - q, u \rangle$ .

Since  $N$  is centrally symmetric,  $\delta_N$  is symmetric and thus a metric. The unit ball under  $\delta_N$  is a convex polyhedron, thus the name polyhedral metric. By construction, an easy calculation shows that for any  $p$  and  $q$  in  $\mathbb{R}^d$ ,  $\|p - q\| \geq \delta_N(p, q) \geq (1 - \epsilon) \cdot \|p - q\|$ . By scaling each vector in  $N$  by a  $1 + \epsilon$  factor, we can ensure that  $(1 + \epsilon) \cdot \|p - q\| \geq \delta_N(p, q) \geq \|p - q\|$ . By taking  $\epsilon$  to be small enough, the optimal clustering for  $\alpha$ -stable clustering instance  $(X, \|\cdot\|, \$)$  is the same as that for  $(X, \delta_N, \$)$  by Lemma 4, and the new instance  $(X, \delta_N, \$)$  is  $(1 - \epsilon)\alpha$ -stable if the original instance  $(X, \|\cdot\|, \$)$  is  $\alpha$ -stable.

**Center proximity.** A clustering instance  $(X, \delta)$  satisfies  $\alpha$ -center proximity property [14] if for any distinct optimal clusters  $X_i$  and  $X_j$  with centers  $c_i$  and  $c_j$  and any point  $p \in X_i$ , one has  $\alpha \cdot \delta(p, c_i) < \delta(p, c_j)$ . Awasthi, Blum, and Sheffet showed that any  $\alpha$ -stable instance satisfies  $\alpha$ -center proximity [14, Fact 2.2] (also [10, Theorem 3.1] under metric perturbation). Optimal solutions of  $\alpha$ -stable instances satisfy the following separation properties.<sup>2</sup>

- $\alpha$ -center proximity implies that  $(\alpha - 1) \cdot \delta(p, c_i) < \delta(p, q)$  for any  $p \in X_i$  and any  $q \notin X_i$ . For  $\alpha \geq 2$ , a point is closer to its own center than to any point of another cluster.<sup>3</sup>
- For  $\alpha \geq 2 + \sqrt{3}$ ,  $\alpha$ -center proximity implies that  $\delta(p, p') < \delta(p, q)$  for any  $p, p' \in X_i$  and any  $q \notin X_i$ . In other words, from any point  $p$  in  $X$ , any *intra*-cluster distance to a point  $p'$  is shorter than any *inter*-cluster distance to a point  $q$ .

We make use of the following stronger intra-inter distance property on  $\alpha$ -stable instances, which allows us to compare *any* intra-distance between two points in  $X_i$  and *any* inter-distance between a point in  $X_i$  and a point in  $X_j$ .

► **Lemma 5.** *Let  $(X, \delta)$  be an  $\alpha$ -stable instance,  $\alpha > 1$ , and let  $X_1$  be a cluster in an optimal clustering with  $q \in X \setminus X_1$  and  $p, p', p'' \in X_1$ . If  $\delta$  is a metric, then  $\delta(p, p') \leq \delta(p'', q)$  for  $\alpha \geq 2 + \sqrt{3}$ . If  $\delta$  is the Euclidean metric in  $\mathbb{R}^d$ , then  $\delta(p, p') \leq \delta(p'', q)$  for  $\alpha \geq 2 + \sqrt{3}$ .*

Finally, we note that it is enough to consider the discrete version of the clustering problem for stable instances.

<sup>2</sup> We give an additional list of known separation properties in the full version of the paper.

<sup>3</sup> They are known as *weak center proximity* [20] and *strict separation property* [18, 22] respectively.

► **Lemma 6.** *For any  $\alpha$ -stable instance  $(X, \delta, \$_\delta)$  with  $\alpha \geq 2 + \sqrt{3}$ , any continuous optimal  $k$ -clustering is a discrete optimal  $k$ -clustering and vice versa.*

### 3 Efficient Dynamic Programming

We now describe a simple, efficient algorithm for computing the optimal clustering for the  $k$ -means,  $k$ -center, and  $k$ -median problem assuming the given instance is  $\alpha$ -stable for  $\alpha \geq 2 + \sqrt{3}$ . Roughly speaking, we make the following observation: if there are at least two clusters, then the two endpoints of the longest edge of the minimum spanning tree of  $X$  belong to different clusters, and no cluster has points in both subtrees of the minimum spanning tree delimited by the longest edge. We describe the dynamic programming algorithm in Section 3.1 and then describe the procedure for computing cluster costs in Section 3.2. We summarize the results in this section by the following theorem.

► **Theorem 7.** *Let  $X$  be a set with  $n$  points lying in  $\mathbb{R}^d$  and  $k \geq 1$  an integer. If the  $k$ -means,  $k$ -median, or  $k$ -center instance for  $X$  under the Euclidean metric is  $\alpha$ -stable for  $\alpha \geq 2 + \sqrt{3} + \varepsilon$  for any constant  $\varepsilon > 0$ , then the optimal clustering can be computed in  $O(nk^2 + n \text{ polylog } n)$  time. For  $d = 2$  the assumption can be relaxed to  $\alpha \geq 2 + \sqrt{3}$ .*

#### 3.1 Fast Dynamic Programming

The following lemma is the key observation for our algorithm.

► **Lemma 8.** *Let  $(X, \delta, \$)$  be an  $\alpha$ -stable  $k$ -clustering instance with  $\alpha \geq 2 + \sqrt{3}$  and  $k \geq 2$ , and let  $T$  be the minimum spanning tree of  $X$  under metric  $\delta$ . Then (1) The two endpoints  $u$  and  $v$  of the longest edge  $e$  in  $T$  do not belong to the same cluster; (2) each cluster lies in the same connected component of  $T \setminus \{e\}$ .*

**Algorithm.** We fix the metric  $\delta$  and the cost function  $\$$ . For a subset  $Y \subseteq X$  and for an integer  $j$  between 1 and  $k - 1$ , let  $\mu(Y; j)$  denote the optimal cost of an  $j$ -clustering on  $Y$  (under  $\delta$  and  $\$$ ). Recall that our definition of  $j$ -clustering required all clusters to be non-empty, so it is not defined for  $|Y| < j$ . For simplicity, we assume that  $\mu(Y; j) = \infty$  for  $|Y| < j$ . Let  $T$  be the minimum spanning tree on  $X$  under  $\delta$ , let  $uv$  be the longest edge in  $T$ ; let  $X_u$  and  $X_v$  be the set of vertices of the two components of  $T \setminus \{uv\}$ . Then  $\mu(X; k)$  satisfies the following recurrence relation:

$$\mu(X; k) = \begin{cases} \mu(X; 1) & \text{if } k = 1, \\ \infty & \text{if } k > |X|, \\ \min_{1 \leq i < k} \{\mu(X_u; i) + \mu(X_v; k - i)\} & \text{if } |X| > 1 \text{ and } k > 1. \end{cases} \quad (1)$$

Using recurrence (1), we compute  $\mu(X; k)$  as follows. Let  $\mathbf{R}$  be a *recursion tree*, a binary tree where each node  $v$  in  $\mathbf{R}$  is associated with a subtree  $T_v$  of  $T$ . If  $v$  is the root of  $\mathbf{R}$ , then  $T_v = T$ . Recursion tree  $\mathbf{R}$  is defined recursively as follows. Let  $X_v \subseteq X$  be the set of vertices of  $T$  in  $T_v$ . If  $|X_v| = 1$ , then  $v$  is a leaf. Each interior node  $v$  of  $\mathbf{R}$  is also associated with the longest edge  $e_v$  of  $T_v$ . Removal of  $e_v$  decomposes  $T_v$  into two connected components, each of which is associated with one of the children of  $v$ . After having computed  $T$ ,  $\mathbf{R}$  can be computed in  $O(n \log n)$  time by sorting the edges in decreasing order of their costs.<sup>4</sup>

<sup>4</sup> Tree  $\mathbf{R}$  is nothing but the minimum spanning tree constructed by Kruskal's algorithm.

For each node  $v \in R$  and for every  $i$  between 1 and  $k - 1$ , we compute  $\mu(X_v; i)$  as follows. If  $v$  is a leaf, we set  $\mu(X_v; 1) = 0$  and  $\mu(X_v; i) = \infty$  otherwise. For all interior nodes  $v$ , we compute  $\mu(X_v; 1)$  using the algorithms described in Section 3.2. Finally, if  $v$  is an interior node and  $i > 1$ , we compute  $\mu(X_v; i)$  using the recurrence relation (1). Recall that if  $w$  and  $z$  are the children of  $v$ , then  $\mu(X_w; \ell)$  and  $\mu(X_z; r)$  for all  $\ell$  and  $r$  have been computed before we compute  $\mu(X_v; i)$ .

Let  $\tau(n)$  be the time spent in computing  $T$  plus the total time spent in computing  $\mu(X_v, 1)$  for all nodes  $v \in R$ . Then the overall time taken by the algorithm is  $O(nk^2 + \tau(n))$ . What is left is to compute the minimum spanning tree  $T$  and all  $\mu(X_v, 1)$  efficiently.

### 3.2 Efficient Implementation

In this section, we show how to obtain the minimum spanning tree and compute  $\mu(X_v; 1)$  efficiently for 1-mean, 1-center, and 1-median when  $X \subseteq \mathbb{R}^d$ . We can compute the Euclidean minimum spanning tree  $T$  in  $O(n \log n)$  time in  $\mathbb{R}^2$  [54]. We can then compute  $\mu(X_v; 1)$  efficiently either under Euclidean metric (for 1-mean), or switch to the  $L_1$ -metric and compute  $\mu(X_v; 1)$  efficiently using Lemma 4 (for 1-center and 1-median).

There are two difficulties in extending the 2D data structures to higher dimensions. No near-linear time algorithm is known for computing the Euclidean minimum spanning tree for  $d \geq 3$ , and we can work with the  $L_1$ -metric only if  $\alpha \geq \sqrt{d}$  (Lemma 4). We address both of these difficulties by working with a polyhedral metric  $\delta_N$ . Let  $\alpha \geq 2 + \sqrt{3} + \Omega(1)$  be the stability parameter. By taking the number of vectors in  $N$  (defined by the polyhedral metric) to be large enough, we can ensure that  $(1 - \epsilon)\|p - q\| \leq \delta_N(p, q) \leq \|p - q\|$  for all  $p, q \in \mathbb{R}^d$ . By Lemma 4,  $X$  is an  $\alpha$ -stable instance under  $\delta_N$  for  $\alpha \geq 2 + \sqrt{3}$ . We first compute the minimum spanning tree of  $X$  in  $O(n \text{ polylog } n)$  time under  $\delta_N$  using the result of Callahan and Kosaraju [24], and then compute  $\mu(X_v, 1)$ .

**Data structure.** We compute  $\mu(X_v; 1)$  in a bottom-up manner. When processing a node  $v$  of  $R$ , we maintain a dynamic data structure  $\Psi_v$  on  $X_v$  from which  $\mu(X_v; 1)$  can be computed quickly. The exact form of  $\Psi_v$  depends on the cost function to be described below. Before that, we analyze the running time  $\tau(n)$  spent on computing every  $\mu(X_v; 1)$ . Let  $w$  and  $z$  be the two children of  $v$ . Suppose we have  $\Psi_w$  and  $\Psi_z$  at our disposal and suppose  $|X_w| \leq |X_z|$ . We insert the points of  $X_w$  into  $\Psi_z$  one by one and obtain  $\Psi_v$  from which we compute  $\mu(X_v; 1)$ . Suppose  $Q(n)$  is the update time of  $\Psi_v$  as well as the time taken to compute  $\mu(X_v; 1)$  from  $\Psi_v$ . The total number of insert operations performed over all nodes of  $R$  is  $O(n \log n)$  because we insert the points of the smaller set into the larger set at each node of  $R$  [42, 53]. Hence  $\tau(n) = O(Q(n) \cdot n \log n)$ . We now describe the data structure for each specific clustering problem.

**1-mean.** We work with the  $L_2$ -metric. Here the center of a single cluster consisting of  $X_v$  is the centroid  $\sigma_v := \left( \sum_{p \in X_v} p \right) / |X_v|$ , and  $\mu(X_v; 1) = \sum_{p \in X_v} \|p\|^2 - |X_v| \cdot \|\sigma_v\|^2$ . At each node  $v$ , we maintain  $\sum_{p \in X_v} p$  and  $\sum_{p \in X_v} \|p\|^2$ . Point insertion takes  $O(1)$  time so  $Q(n) = 1$ .

**1-center.** As mentioned in the beginning of the section, we can work with the  $L_1$ -metric for  $d = 2$ . We wish to find the smallest  $L_1$ -disc (a diamond) that contains  $X_v$ . Let  $e^+ = (1, 1)$  and  $e^- = (-1, 1)$ . Then the radius  $\rho_v$  of the smaller  $L_1$ -disc containing  $X_v$  is

$$\rho_v = \frac{1}{2} \max \left\{ \max_{p \in X_v} \langle p, e^+ \rangle - \min_{p \in X_v} \langle p, e^+ \rangle, \max_{p \in X_v} \langle p, e^- \rangle - \min_{p \in X_v} \langle p, e^- \rangle \right\}. \quad (2)$$

We maintain the following four terms  $\max_{p \in X_v} \langle p, e^+ \rangle$ ,  $\min_{p \in X_v} \langle p, e^+ \rangle$ ,  $\max_{p \in X_v} \langle p, e^- \rangle$ , and  $\min_{p \in X_v} \langle p, e^- \rangle$  at  $v$ . A point can be inserted in  $O(1)$  time and  $\rho_v$  can be computed from these four terms in  $O(1)$  time. Therefore,  $Q(n) = O(1)$ . For  $d > 2$ , we work with a polyhedral metric and compute the smallest ball  $B(X_v)$  that contains  $X_v$ . For full details, see the full version of the paper.

**1-median.** Similar to 1-center, we work with the polyhedral metric. Fix a node  $v$  of  $T$ . For a point  $x \in \mathbb{R}^d$ , let  $F_v(x) = \sum_{p \in X_v} \delta_N(x, p)$  which is a piecewise-linear function. Our goal is to compute  $\xi_v^* = \arg \min_{x \in \mathbb{R}^d} F_v(x)$ . Our data structure is a dynamic range-tree [3] used for orthogonal range searching that can insert a point in  $O(\log n)$  time. Using multi-dimensional parametric search [5],  $\xi_v^*$  can be computed in  $O(\text{poly log } n)$  time after each update; see the full version of the paper for details.

#### 4 $k$ -Median: Single-Swap Local Search

We customize the standard local-search framework for the  $k$ -clustering problem [30, 31, 39]. In order to recover the optimal solution, we must define near-optimality more carefully. Let  $(X, \delta)$  be an instance of  $\alpha$ -stable  $k$ -median in  $\mathbb{R}^2$  for  $\alpha > 5$ . By Lemma 6, it suffices to consider the *discrete*  $k$ -median problem. In Section 4, we describe a simple local-search algorithm for finding the optimal clustering of  $(X, \delta)$ . In Section 4 we show that the algorithm terminates within  $O(k \log(n\Delta))$  iterations. We obtain the following.

► **Theorem 9.** *Let  $(X, \delta)$  be an  $\alpha$ -stable instance of the  $k$ -median problem for some  $\alpha > 5$  where  $X$  is a set of  $n$  points in  $\mathbb{R}^2$  equipped with  $L_p$ -metric  $\delta$ . The 1-swap local search algorithm terminates with the optimal clustering in  $O(k \log(n\Delta))$  iterations.*

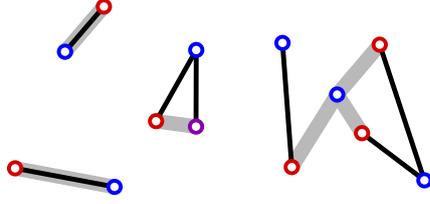
**Local-search algorithm.** The local-search algorithm maintains a  $k$ -clustering induced by a set  $S$  of  $k$  cluster centers. At each step, it finds a pair of points  $x \in X$  and  $y \in S$  such that  $\$(X, S + x - y)$  is minimized. If  $\$(X, S + x - y) \geq \$(X, S)$ , it stops and returns the  $k$ -clustering induced by  $S$ . Otherwise it replaces  $S$  with  $S + x - y$  and repeats the above step. The pair  $(x, y)$  will be referred to as a *1-swap*.

**Local-search analysis.** The high-level structure of our analysis follows Friggstad et al. [39], however new ideas are needed for 1-swap. In this subsection, we denote a  $k$ -clustering by the set of its cluster centers. Let  $S$  be a fixed  $k$ -clustering, and let  $O$  be the optimal clustering. For a subset  $Y \subseteq X$ , we use  $\$(Y)$  and  $\$(Y)$  to denote  $\$(Y, S)$  and  $\$(Y, O)$ , respectively. Similarly, for a point  $p \in X$ , we use  $\text{nn}(p)$  and  $\text{nn}^*(p)$  to denote the nearest neighbor of  $p$  in  $S$  and in  $O$ , respectively; define  $\delta(p)$  to be  $\delta(p, S)$  and  $\delta^*(p)$  to be  $\delta(p, O)$ . We partition  $X$  into four subsets as follows:

- $X_{00} := \{p \in X \mid \text{nn}(p) \in S \setminus O, \text{nn}^*(p) \in O \setminus S\}$ ;
- $X_{01} := \{p \in X \mid \text{nn}(p) \in S \setminus O, \text{nn}^*(p) \in S \cap O\}$ ;
- $X_{10} := \{p \in X \mid \text{nn}(p) \in S \cap O, \text{nn}^*(p) \in O \setminus S\}$ ;
- $X_{11} := \{p \in X \mid \text{nn}(p) \in S \cap O, \text{nn}^*(p) \in S \cap O\}$ .

Observe that for any point  $p$  in  $X_{11}$ ,  $\text{nn}(p) = \text{nn}^*(p)$  and  $\$(p) = \$(p)$ ; for any point  $p$  in  $X_{01}$ , one has  $\$(p) \leq \$(p)$ ; and for any point  $p$  in  $X_{10}$ , one has  $\$(p) \geq \$(p)$ . Costs  $\delta(p)$  and  $\delta^*(p)$  are not directly comparable for point  $p$  in  $X_{00}$ . A  $k$ -clustering  $S$  is  *$C$ -good* for some parameter  $C \geq 0$  if  $\$(X) \leq \$(X) + C \cdot \$(X_{00})$ .

► **Lemma 10.** *Any  $C$ -good clustering  $S$  for an  $\alpha$ -stable clustering instance  $(X, \delta, \$)$  must be optimal for  $\alpha \geq C + 1$ .*



■ **Figure 1** Illustration of candidate swaps  $\mathcal{S}$  in  $\mathbb{R}^2$ . The blue dots belong to set  $S$ , the red dots belong to set  $O$ ; the only purple dot is in  $S \cap O$ . The thick gray segments indicate pairs inside the stars; each star has exact one blue dot as its center. The black pairs are the candidate swaps. Notice that the partitions of  $S$  and  $O$  form connected components.

**Proof.** Define a perturbed distance function  $\tilde{\delta}: X \times X \rightarrow \mathbb{R}_{\geq 0}$  with respect to the given clustering  $S$  as follows:

$$\tilde{\delta}(p', p) := \begin{cases} \alpha \cdot \delta(p', p) & \text{if } p \neq \text{nn}(p'), \\ \delta(p', p) & \text{otherwise.} \end{cases}$$

Note that  $\tilde{\delta}$  is not symmetric. Let  $\tilde{\mathbb{S}}(\cdot, \cdot)$  denote the cost function under the perturbed distance function  $\tilde{\delta}$ . The optimal clustering under perturbed cost function is the same as the original optimal clustering  $O$  by the stability assumption. Since  $\text{nn}(p) = \text{nn}^*(p)$  if and only if  $p \in X_{11}$ , the cost of  $O$  under the perturbed cost can be written as:

$$\tilde{\mathbb{S}}(X, O) = \alpha \cdot \mathbb{S}(X_{00}, O) + \alpha \cdot \mathbb{S}(X_{01}, O) + \alpha \cdot \mathbb{S}(X_{10}, O) + \mathbb{S}(X_{11}, O).$$

By definition of perturbed distance  $\tilde{\delta}$ ,  $\tilde{\mathbb{S}}(X, S) = \mathbb{S}(X, S)$ . Now, by the assumption that clustering  $S$  is  $C$ -good,

$$\begin{aligned} \tilde{\mathbb{S}}(X, S) = \mathbb{S}(X, S) &\leq \mathbb{S}(X, O) + C \cdot \mathbb{S}(X_{00}, O) \\ &\leq (C + 1) \cdot \mathbb{S}(X_{00}, O) + \mathbb{S}(X_{01}, O) + \mathbb{S}(X_{10}, O) + \mathbb{S}(X_{11}, O) \\ &\leq \tilde{\mathbb{S}}(X, O); \end{aligned}$$

the last inequality follows by taking  $\alpha \geq C + 1$ . This implies that  $S$  is an optimal clustering for  $(X, \tilde{\delta})$ , and thus is equal to  $O$ . ◀

Next, we prove a lower bound on the improvement in the cost of a clustering that is not  $C$ -good after performing a 1-swap. Following Arya et al. [12], define the set of *candidate swaps*  $\mathcal{S}$  as follows: For each center  $i$  in  $S$ , consider the *star*  $\Sigma_i$  centered at  $i$  defined as the collection of pairs  $\Sigma_i := \{(i, j) \in S \times O \mid \text{nn}(j) = i\}$ . Denote  $\text{center}(j)$  to be the center of the star where  $j$  belongs; in other words,  $\text{center}(j) = i$  if  $j$  belongs to  $\Sigma_i$ .

For  $i \in S$ , let  $O_i := \{j \in O \mid \text{center}(j) = i\}$  be the set of centers of  $O$  in star  $\Sigma_i$ . If  $|O_i| = 1$ , then we add the only pair  $(i, j) \in \Sigma_i$  to the candidate set  $\mathcal{S}$ . Let  $S_\emptyset := \{i \in S \mid O_i = \emptyset\}$ . Let  $O_{>1}$  contain centers in  $O$  that belong to a star of size greater than 1. We pick  $|O_{>1}|$  pairs from  $S_\emptyset \times O_{>1}$  such that each point of  $O_{>1}$  is matched only once and each point of  $S_\emptyset$  is matched at most twice and add them to  $\mathcal{S}$ ; this is feasible because  $|S_\emptyset| \geq |O_{>1}|/2$ . Since each center in  $O$  belongs to exactly one pair of  $\mathcal{S}$ ,  $|\mathcal{S}| = k$ . By construction, if  $|\Sigma_i| \geq 2$ , then  $i$  does not belong to any candidate swap. See Figure 1.

▶ **Lemma 11.** *For each point  $p$  in  $X_{01}$ ,  $X_{10}$ , or  $X_{11}$ , the set of candidate swaps  $\mathcal{S}$  satisfies*

$$\sum_{(i,j) \in \mathcal{S}} (\delta(p) - \delta'(p)) \geq \delta(p) - \delta^*(p); \tag{3}$$

and for each point  $p$  in  $X_{00}$ , the set of candidate swaps  $\mathcal{S}$  satisfies

$$\sum_{(i,j) \in \mathcal{S}} (\delta(p) - \delta'(p)) \geq (\delta(p) - \delta^*(p)) - 4\delta^*(p), \quad (4)$$

where  $\$'$  is the cost function on  $X$  defined with respect to  $S' := S - i + j$ , and  $\delta'(p)$  is the distance between  $p$  and its nearest neighbor in  $S'$ .

**Proof.** For point  $p$  in  $X_{11}$ , both  $\text{nn}(p)$  and  $\text{nn}^*(p)$  are in  $S'$ , so  $\delta'(p) = \delta(p) = \delta^*(p)$ . For point  $p$  in  $X_{01}$ ,  $\delta(p) \leq \delta^*(p)$ ; when  $\text{nn}(p)$  is being swapped out by some in 1-swap  $S'$ ,  $\text{nn}^*(p)$  must be in  $S'$ . For point  $p$  in  $X_{10}$ ,  $\delta(p) \geq \delta^*(p)$ ; center  $\text{nn}(p)$  will never be swapped out by any 1-swap in  $\mathcal{S}$ , so  $\delta'(p) \leq \delta(p)$ . By construction of  $\mathcal{S}$ , there is exactly one choice of  $S'$  that swaps  $\text{nn}^*(p)$  in; for that particular swap we have  $\delta'(p) = \delta^*(p)$ . In all three cases one has inequality (3). Our final goal is to prove inequality (4). Consider a swap  $(i, j)$  in  $\mathcal{S}$ . There are three cases to consider:

- $j = \text{nn}^*(p)$ . There is exactly one swap for which  $j = \text{nn}^*(p)$ . In this case  $\delta(p) \leq \delta^*(p)$ , therefore  $\delta(p) - \delta'(p) \geq \delta(p) - \delta^*(p)$ .
- $j \neq \text{nn}^*(p)$  and  $i \neq \text{nn}(p)$ . Since  $\text{nn}(p) \in S'$ ,  $\delta'(p) \leq \delta(p)$ . Therefore  $\delta(p) - \delta'(p) \geq 0$ .
- $j \neq \text{nn}^*(p)$  and  $i = \text{nn}(p)$ . By construction, there are most two swaps in  $\mathcal{S}$  that may swap out  $\text{nn}(p)$ . We claim that  $i \neq \text{center}(\text{nn}^*(p))$ . Indeed, if  $i = \text{center}(\text{nn}^*(p))$ , then by construction,  $\Sigma_i = \{(i, \text{nn}^*(p))\}$  because the center of star of size greater than one is never added to a candidate swap. But this contradicts the assumption that  $j \neq \text{nn}^*(p)$ . The claim implies that  $\text{center}(\text{nn}^*(p)) \in S'$  and thus  $\delta'(p) \leq \delta(p, \text{center}(\text{nn}^*(p)))$ . We obtain a bound on  $\delta(p, \text{center}(\text{nn}^*(p)))$  as follows:

$$\begin{aligned} \delta(p, \text{center}(\text{nn}^*(p))) &\leq \delta(p, \text{nn}^*(p)) + \delta(\text{nn}^*(p), \text{center}(\text{nn}^*(p))) \\ &\leq \delta^*(p) + \delta(\text{nn}^*(p), \text{nn}(p)) \\ &\leq \delta^*(p) + (\delta^*(p) + \delta(p)) = \delta(p) + 2\delta^*(p). \end{aligned}$$

Therefore,  $\delta(p) - \delta'(p) \geq \delta(p) - \delta(p, \text{center}(\text{nn}^*(p)))$ . Putting everything together, we obtain:

$$\sum_{S' \in \mathcal{S}} (\delta(p) - \delta'(p)) \geq (\delta(p) - \delta^*(p)) + 0 + 2(\delta(p) - \delta(p) - 2\delta^*(p)) = \delta(p) - 5\delta^*(p).$$

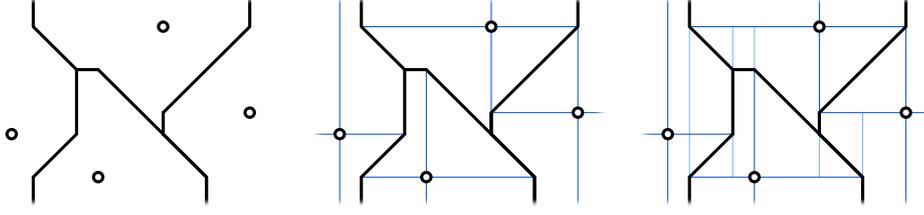
Using Lemma 11, we can prove the following.

► **Lemma 12.** *Let  $S$  be a  $k$ -clustering of  $(X, \delta)$  that is not  $C$ -good for some fixed constant  $C > 4 + \epsilon$  with arbitrarily small  $\epsilon > 0$ . There is always a 1-swap  $S'$  such that  $\$(X) - \$(X) \leq (1 - \epsilon/(1 + \epsilon)k) \cdot (\$(X) - \$(X))$ , where  $\$'$  is the cost function defined with respect to  $S'$ .*

**Proof.** By Lemma 11 one has  $\$(X) - \$(X) \geq (\$(X) - \$(X) - \Psi(X_{00}))/k$  for some 1-swap  $S'$  and its corresponding cost function  $\$(\cdot)$ . Since  $S$  is not  $C$ -good,  $\$(X) - \$(X) > C \cdot \$(X_{00})$ . Rearranging and plugging the definition of  $\Psi(\cdot)$ , we have

$$\begin{aligned} \$(X) - \$(X) &\leq \$(X) - \$(X) - (\$(X) - \$(X) - \Psi(X_{00}))/k \\ &\leq \$(X) - \$(X) - (\$(X) - \$(X) - 4 \cdot \$(X_{00}))/k \\ &\leq \$(X) - \$(X) \\ &\quad - (\$(X) - \$(X) + (M - 1) \cdot (\$(X) - \$(X)) - 4M \cdot \$(X_{00}))/Mk \\ &\leq \left(1 - \frac{\epsilon}{(1 + \epsilon)k}\right) \cdot (\$(X) - \$(X)), \end{aligned}$$

where the last inequality holds by taking  $M$  to be arbitrarily large (say  $M > 1 + 1/\epsilon$ ). ◀



■ **Figure 2**  $L_1$  Voronoi diagram  $V$ , quadrant decomposition  $\tilde{V}$ , and trapezoid decomposition  $V^\parallel$ .

## 5 Efficient Implementation of Local Search

We describe an efficient implementation of each step of the local-search algorithm in this section. By Lemma 4, it suffices to implement the algorithm using a polyhedral metric  $\delta_N$ . We show that each step of 1-swap can be implemented in  $O(nk^{2d-1} \text{polylog } n)$  time under the assumption that  $\alpha > 5$ . We obtain the following:

► **Theorem 13.** *Let  $(X, \delta)$  be an  $\alpha$ -stable instance of the  $k$ -median problem where  $X \subset \mathbb{R}^d$  and  $\delta$  is the Euclidean metric. For  $\alpha > 5$ , the 1-swap local search algorithm computes the optimal  $k$ -clustering of  $(X, \delta)$  in  $O(nk^{2d-1} \text{polylog } n)$  time.*

For simplicity, we present a slightly weaker result for  $d = 2$  using the  $L_1$ -metric, as it is straightforward to implement and more intuitive. Using the  $L_1$ -metric requires  $\alpha > 5\sqrt{2}$ . The extension to higher dimensional Euclidean space using the polyhedral metric is described in the full version of the paper, which works for  $\alpha > 5$ .

**Voronoi diagram under  $L_1$  norm.** First, we fix a point  $x \in X \setminus S$  to insert and a center  $y \in S$  to drop. Define  $S' := S + x - y$ . We build the  $L_1$  Voronoi diagram  $V$  of  $S'$ . The cells of  $V$  may not be convex, but they are *star-shaped*: for any  $c \in S'$  and for any point  $x \in \text{Vor}(c)$ , the segment  $cx$  lies completely in  $\text{Vor}(c)$ . Furthermore, all line segments on the cell boundaries of  $V$  must have slopes belonging to one of the four possible values: vertical, horizontal, diagonal, or antidiagonal.

Next, decompose each Voronoi cell  $\text{Vor}(c)$  into four *quadrants* centered at  $c$ . Denote the resulting subdivision of  $V$  as  $\tilde{V}$ . We compute a *trapezoidal decomposition*  $V^\parallel$  of the diagram  $\tilde{V}$  by drawing a vertical segment from each vertex of  $\tilde{V}$  in both directions until it meets an edge of  $V$ ;  $V^\parallel$  has  $O(k)$  trapezoids, see Figure 2. For each trapezoid  $\tau \in V^\parallel$ , let  $X_\tau := X \cap \tau$ . The cost of the new clustering  $S'$  can be computed as  $\$(X, S') = \sum_{\tau \in V^\parallel} \$(X_\tau, S')$ .

**Range-sum queries.** Now we discuss how to compute  $\$(X_\tau, S')$ . Each trapezoid  $\tau$  in cells  $\text{Vor}(c)$  is associated with a vector  $u(\tau) \in \{\pm 1\}^2$ , depending on which of the four quadrants  $\tau$  belongs to with respect to the axis-parallel segments drawn passing through the center  $c$  of the cell. If  $\tau$  lies in the top-right quadrant then  $u(\tau) = (1, 1)$ . Similarly if  $\tau$  lies in the top-left (resp. bottom-left, bottom-right) then  $u(\tau) = (-1, 1)$  (resp.  $(-1, -1)$ ,  $(1, -1)$ ).

$$\$(X_\tau, S') = \sum_{x \in X_\tau} \|x - c\|_1 = \sum_{x \in X_\tau} \langle x - c, u(\tau) \rangle = \sum_{x \in X_\tau} \langle x, u(\tau) \rangle - |X_\tau| \cdot \langle c, u(\tau) \rangle. \quad (5)$$

We preprocess  $X$  into a data structure that answers the following query:

- **TRAPEZOIDSUM** $(\tau, u)$ : Given a trapezoid  $\tau$  and a vector  $u \in \{\pm 1\}^2$ , return  $|X \cap \tau|$  as well as  $\sum_{x \in X \cap \tau} \langle x, u \rangle$ .

<p><b>1-SWAP(<math>X, S</math>):</b>  <i>input:</i> Point set <math>X</math> and centers <math>S</math>  for each point <math>x \in X \setminus S</math> and center <math>y \in S</math>:  <math>S' \leftarrow S + x - y</math>  <math>V \leftarrow L_1</math> Voronoi diagram of <math>S'</math>  <math>\tilde{V} \leftarrow</math> decompose each cell <math>\text{Vor}(c)</math> into four quadrants centered at <math>c</math>  <math>V^\parallel \leftarrow</math> trapezoidal decomposition of <math>\tilde{V}</math>  for each trapezoid <math>\tau \in V^\parallel</math>:  <math>\\$(X_\tau, S') \leftarrow \text{TRAPEZOIDSUM}(\tau, u(\tau))</math>  <math>\\$(X, S') \leftarrow \sum_{\tau \in V^\parallel} \\$(X_\tau, S')</math>  return <math>(x, y)</math> with the lowest <math>\\$(X, S + x - y)</math></p>
---

■ **Figure 3** Efficient implementation of 1-swap under 1-norm.

The above query can be viewed as a 3-oriented polygonal range query [33]. We construct a 3-level range tree  $\Psi$  on  $X$ . Omitting the details (which can be found in [33]),  $\Psi$  can be constructed in  $O(n \log^2 n)$  time and uses  $O(n \log^2 n)$  space. Each node  $\xi$  at the third level of  $\Psi$  is associated with a subset  $X_\xi \subseteq X$ . We store  $w(\xi, u) := \sum_{x \in X_\xi} \langle x, u \rangle$  for each  $u \in \{\pm 1\}^2$  and  $|X_\xi|$  at  $\xi$ . For a trapezoid  $\tau$ , the query procedure identifies in  $O(\log^3 n)$  time a set  $\Xi_\tau$  of  $O(\log^3 n)$  third-level nodes such that  $X \cap \tau = \cup_{\xi \in \Xi_\tau} X_\xi$  and each point of  $X \cap \tau$  appears as exactly one node of  $\Xi_\tau$ . Then  $\sum_{x \in X_\tau} \langle x, u \rangle = \sum_{\xi \in \Xi_\tau} w(\xi, u)$  and  $|X_\tau| = \sum_{\xi \in \Xi_\tau} |X_\xi|$ .

With the information stored at the nodes in  $\Xi_\tau$ ,  $\text{TRAPEZOIDSUM}(\tau, u)$  query can be answered in  $O(\log^3 n)$  time. By performing  $\text{TRAPEZOIDSUM}(\tau, u(\tau))$  query for all  $\tau \in V^\parallel$ ,  $\$(X_\tau, S')$  can be computed in  $O(k \log^3 n)$  time since  $V^\parallel$  has a total of  $O(k)$  trapezoids.

We summarize the implementation of 1-swap algorithm in Figure 3. The 1-swap procedure considers at most  $nk$  different  $k$ -clustering. Therefore we obtain the following.

► **Lemma 14.** *Let  $(X, \delta, \$)$  be a given clustering instance where  $\delta$  is the  $L_1$  metric, and let  $S$  be a given  $k$ -clustering. After  $O(n \log n)$  time preprocessing, we find a  $k$ -clustering  $S' := S + x - y$  minimizing  $\$(X, S')$  among all choices of  $(x, y)$  in  $O(nk^2 \log^3 n)$  time.*

## 6 Conclusion

We presented near-linear time algorithms for finding optimal solutions of stable clustering instances for the  $k$ -means,  $k$ -medians, and  $k$ -center problem. We note that variants of all three approaches might work for smaller values of  $\alpha$ . The value of  $\alpha$  assumed in our results is larger than what is known for polynomial-time algorithm (e.g.  $\alpha \geq 2$  in Angelidakis et al. [10]) and that in some applications the input may not satisfy our assumption, but our results are a big first step toward developing near-linear time algorithms for stable instances. We are not aware of any previous near-linear time algorithms for computing optimal clustering even for larger values of  $\alpha$ . We leave the problem of reducing the assumption on  $\alpha$  as an important open question.

---

## References

- 1 Margareta Ackerman and Shai Ben-David. Clusterability: A theoretical study. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, volume 5 of *JMLR Proceedings*, pages 1–8, 2009.
- 2 Peyman Afshani, Jérémy Barbay, and Timothy M Chan. Instance-optimal geometric algorithms. *Journal of the ACM (JACM)*, 64(1):3, 2017.

- 3 Pankaj K Agarwal, Jeff Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.
- 4 Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.
- 5 Pankaj K Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.
- 6 Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Approximate clustering with same-cluster queries. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:21, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.40.
- 7 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Papat. NP-hardness of Euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.
- 8 Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 429–437. ACM, 2017.
- 9 Haris Angelidakis, Pranjali Awasthi, Avrim Blum, Vaggos Chatziafratis, and Chen Dan. Bilinial stability, certified algorithms and the independent set problem. Preprint, October 2018.
- 10 Haris Angelidakis, Konstantin Makarychev, and Yury Makarychev. Algorithms for stable and perturbation-resilient problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 438–451. ACM, 2017.
- 11 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean  $k$ -medians and related problems. In *STOC*, volume 98, pages 106–113, 1998.
- 12 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for  $k$ -median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, January 2004. doi:10.1137/S0097539702416402.
- 13 Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. In *Advances in neural information processing systems*, pages 3216–3224, 2016.
- 14 Pranjali Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1–2):49–54, 2012.
- 15 Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 250–257. ACM, 2002.
- 16 Ainesh Bakshi and Nadiia Chepurko. Polynomial time algorithm for 2-stable clustering instances. Preprint, July 2016.
- 17 Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Approximate clustering without the approximation. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1068–1077. Society for Industrial and Applied Mathematics, 2009.
- 18 Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 671–680. ACM, 2008.
- 19 Maria-Florina Balcan and Mark Braverman. Nash equilibria in perturbation-stable games. *Theory of Computing*, 13(1):1–31, 2017.
- 20 Maria-Florina Balcan, Nika Haghtalab, and Colin White.  $k$ -center clustering under perturbation resilience. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2016.68.
- 21 Maria Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

- 22 Shalev Ben-David and Lev Reyzin. Data stability in clustering: A closer look. *Theoretical Computer Science*, 558(1):51–61, 2014.
- 23 Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(5):643–660, 2012.
- 24 Paul B Callahan and S Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 291–300. Society for Industrial and Applied Mathematics, 1993.
- 25 Chandra Chekuri and Shalmoli Gupta. Perturbation resilient clustering for  $k$ -center and related problems via LP relaxations. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.9.
- 26 Vincent Cohen-Addad. A fast approximation scheme for low-dimensional  $k$ -means. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 430–440, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3174304.3175298>.
- 27 Vincent Cohen-Addad, Arnaud de Mesmay, Eva Rotenberg, and Alan Roytman. The bane of low-dimensionality clustering. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 441–456, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3174304.3175300>.
- 28 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saupic. Near-linear time approximation schemes for clustering in doubling metrics. Preprint, June 2019.
- 29 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for  $k$ -Median and  $k$ -Means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- 30 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for  $k$ -means and  $k$ -median in Euclidean and minor-free metrics. *SIAM Journal on Computing*, 48(2):644–667, 2019.
- 31 Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 49–60. IEEE, 2017.
- 32 Sanjoy Dasgupta. The hardness of  $k$ -means clustering. Technical report, Department of Computer Science and Engineering, University of California, September 2008.
- 33 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.
- 34 Amit Deshpande, Anand Louis, and Apoorv Vikram Singh. On Euclidean  $k$ -means clustering with  $\alpha$ -center proximity. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2087–2095, 2019.
- 35 Abhratanu Dutta, Aravindan Vijayaraghavan, and Alex Wang. Clustering stable instances of Euclidean  $k$ -means. In *Advances in Neural Information Processing Systems*, pages 6500–6509, 2017.
- 36 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444. ACM, 1988.
- 37 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A ptas for  $k$ -means clustering based on weak coresets. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 11–18. ACM, 2007.

- 38 Zachary Friggstad, Kamyar Khodamoradi, and Mohammad R. Salavatipour. Exact algorithms and lower bounds for stable instances of Euclidean  $k$ -means. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2958–2972, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3310435.3310618>.
- 39 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for  $k$ -means in doubling metrics. *SIAM Journal on Computing*, 48(2):452–480, 2019. doi:10.1137/17M1127181.
- 40 Sarel Har-Peled. No, coresets, no cry. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 324–335. Springer, 2004.
- 41 Sarel Har-Peled and Soham Mazumdar. On coresets for  $k$ -means and  $k$ -median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2004.
- 42 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
- 43 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the  $k$ -center problem. *Math. Oper. Res.*, 10(2):180–184, May 1985.
- 44 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 731–740. ACM, 2002.
- 45 Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the  $k$ -means algorithm. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 299–308. IEEE, 2010.
- 46 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time  $(1 + \epsilon)$ -approximation algorithm for  $k$ -means clustering in any dimensions. In *Annual Symposium on Foundations of Computer Science*, volume 45, pages 454–462. IEEE COMPUTER SOCIETY PRESS, 2004.
- 47 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for  $k$ -means. *Information Processing Letters*, 120:40–43, 2017.
- 48 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar  $k$ -means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012.
- 49 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu-Linial stable instances of max cut and minimum multiway cut. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 890–906. SIAM, 2014.
- 50 Nimrod Megiddo and Kenneth J Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.
- 51 Matúš Mihalák, Marcel Schöngens, Rastislav Šrámek, and Peter Widmayer. On the complexity of the metric TSP under stability considerations. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 382–393. Springer, 2011.
- 52 Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of Lloyd-type methods for the  $k$ -means problem. *Journal of the ACM (JACM)*, 59(6):28, 2012.
- 53 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.
- 54 Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman, editors. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.

# Width Notions for Ordering-Related Problems

**Emmanuel Arrighi** 

University of Bergen, Norway  
emmanuel.arrighi@uib.no

**Henning Fernau** 

University of Trier, Germany  
fernau@uni-trier.de

**Mateus de Oliveira Oliveira** 

University of Bergen, Norway  
mateus.oliveira@uib.no

**Petra Wolf** 

University of Trier, Germany  
wolfp@informatik.uni-trier.de

---

## Abstract

We are studying a weighted version of a linear extension problem, given some finite partial order  $\rho$ , called COMPLETION OF AN ORDERING. While this problem is NP-complete, we show that it lies in FPT when parameterized by the interval width of  $\rho$ . This ordering problem can be used to model several ordering problems stemming from diverse application areas, such as graph drawing, computational social choice, or computer memory management. Each application yields a special  $\rho$ . We also relate the interval width of  $\rho$  to parameterizations such as *maximum range* that have been introduced earlier in these applications, sometimes improving on parameterized algorithms that have been developed for these parameterizations before. This approach also gives some practical sub-exponential time algorithms for ordering problems.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Fixed parameter tractability; Theory of computation  $\rightarrow$  Dynamic programming; Mathematics of computing  $\rightarrow$  Combinatorial optimization

**Keywords and phrases** Parameterized algorithms, interval width, linear extension, one-sided crossing minimization, Kemeny rank aggregation, grouping by swapping

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.9

**Funding** *Emmanuel Arrighi*: Research Council of Norway (Grant no. 274526), IS-DAAD (Grant no. 309319).

*Henning Fernau*: DAAD PPP (Grant no. 57525246).

*Mateus de Oliveira Oliveira*: Trond Mohn Foundation, Research Council of Norway (Grant no. 288761), IS-DAAD (Grant no. 309319).

*Petra Wolf*: DFG project FE 560/9-1, DAAD PPP (Grant no. 57525246).

## 1 Introduction

Many computational problems can be phrased as the task of arranging a collection of combinatorial objects into a minimum-cost linear order that satisfies certain constraints. Examples of natural problems that fall in this category are ONE-SIDED CROSSING MINIMIZATION (OSCM), a prominent problem in the field of graph drawing and VLSI design [4, 32, 45, 50, 52], GROUPING BY SWAPPING (GBS), a problem with applications in computer memory management [15, 28, 55], and KEMENY RANK AGGREGATION (KRA), a prominent problem in the field of computational social choice [19, 36]. A natural parameter that arises when studying problems such as OSCM, GBS and KRA from the perspective of parameterized complexity theory is the cost  $k$  of a solution. In particular, the best algorithm



© Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, and Petra Wolf; licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 9; pp. 9:1–9:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for OSCM, parameterized by the cost of a solution  $k$ , is the algorithm due to Kobayashi and Tamaki [38] which runs in time<sup>1</sup>  $\mathcal{O}^*(2^{\sqrt{2k}})$  and the best single-exponential algorithm for KRA runs in time  $\mathcal{O}^*(1.403^k)$  [51], while sub-exponential algorithms of type  $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$  have been proposed in [35], with some unclear constant hidden in the  $\mathcal{O}$ -notation of the exponent. Not surprisingly, they have been devised with substantially distinct sets of techniques.

In this paper, significantly extending the ideas started out in [23, 25], we leverage the COMPLETION OF AN ORDERING problem (CO) to provide a unified framework for the study of several cost-parameterized ordering problems. In this problem, we are given a partial order  $\rho$  on a set  $V$ , and a function  $c : V \times V \rightarrow \mathbb{N}$  assigning costs to incomparable pairs, and the goal is to compute a minimum-cost linear extension of  $\rho$ . Interestingly, a natural *structural* parameter that arises in this context is the pathwidth of the cocomparability graph of the input partial order  $\rho$ . This graph has  $V$  as vertex-set and there is an undirected edge between vertices  $v$  and  $v'$  if and only if  $v$  and  $v'$  are not related in the partial order. Our main result states that CO, parameterized by the interval width  $w$  of the input partial order, can be solved in time  $\mathcal{O}^*(2^w)$ . Additionally, our algorithm is optimal under ETH. Using our main result, together with reductions from OSCM, GBS and KRA to PCO, the natural restriction of CO to positive costs, we obtain algorithms for these three problems (parameterized by width, or by the standard parameter, or by other problem-specific structural parameters) whose running times often match or improve on the best algorithms for the three problems.

When reducing OSCM or GBS to PCO, the partial order one obtains is an interval order, meaning that the cocomparability graph of this order is an interval graph. Interval orders play an important role in partial order theory due to the fact that their interval width can be computed in linear time. Additionally, they find applications in many contexts of practical relevance such as scheduling, online and packing algorithms, see [54]. Inspired by this, we define the POSITIVE COMPLETION OF AN INTERVAL ORDERING (PCIO) problem, a version of PCO where the input partial order is required to be an interval order. In this restricted version, our main algorithm for CO parameterized by interval width can be converted into a sub-exponential  $\mathcal{O}^*(2^{\sqrt{2k}})$ -time FPT algorithm for PCIO, parameterized by cost  $k$ .

Our width-based approach also allows us to improve on a parameterized algorithm for KRA based on the parameter *maximum range* (of a candidate) as introduced and discussed in [5]. Further, it can be used to show that GBS is also fixed parameter tractable when parameterized by a parameter called scope coincidence degree, a natural parameter in the context of strings. This gives the first algorithmic use of this structural string parameter.

Our approach for CO is built on dynamic programming on a path decomposition of the cocomparability graph of the partial order. Notice that this path decomposition structure has been recently exploited for counting the number of linear extensions by Eiben *et al.* [22]. Here, we use this approach to find the cheapest linear extension.

## 2 Preliminaries

In this section, we collect the basic notions of this paper.  $\mathbb{N}$  denotes the set of non-negative integers and  $\mathbb{N}_{>0}$  denotes the set of positive integers. Given  $r \in \mathbb{N}_{>0}$ , we write  $[r] \doteq \{1, \dots, r\}$ .

**Notation on Partial Orders.** Let  $V$  be a set. A *partial order* over  $V$  is a reflexive, anti-symmetric and transitive binary relation  $\rho \subseteq V \times V$ . We say that  $\rho$  is a *linear order* if additionally, for each  $(x, y) \in V \times V$ , either  $(x, y) \in \rho$  or  $(y, x) \in \rho$ . A *strict partial order*

<sup>1</sup> Recall that the  $\mathcal{O}^*$ -notation suppresses polynomial factors.

over  $V$  is an irreflexive and transitive binary relation  $\sigma \subseteq V \times V$ . By adding the identity relation  $I_V$ ,  $\rho \doteq \sigma \cup I_V$  becomes a partial order, and conversely from a partial order  $\rho$  on  $V$ , we can define  $\sigma \doteq \rho \setminus I_V$  as a strict partial order. Hence, we will occasionally use the term linear order also for the corresponding strict order, often denoted as  $<_\rho$  for reasons of clarity. Notice that for finite base sets  $V$ , we can specify a linear order  $<_f$  by a bijection  $f : [|V|] \rightarrow V$ , with the understanding that  $f(i) <_f f(j)$  if and only if  $i < j$ , i.e., if the number  $i$  is smaller than the number  $j$ . Such a bijection  $f$  is also called a *ranking* in the following. Conversely, any linear order  $\tau$  on  $\Sigma$  defines a bijection  $f_\tau : [|V|] \rightarrow V$ .

Given two partial orders  $\rho, \tau \subseteq V \times V$ ,  $\tau$  is an *extension* of  $\rho$  if  $\rho \subseteq \tau$ . If  $\tau$  is also a linear order on  $V$ , then  $\tau$  is a *linear extension* of  $\rho$ . Given a linear order  $\tau$  on  $V$ , let  $\min_\tau(V)$  be the minimum element in  $V$  with respect to  $\tau$  and  $\max_\tau(V)$  be the maximum element in  $V$  with respect to  $\tau$ . Given a subset  $T \subseteq V$  and a partial order  $\rho \subseteq V \times V$ , let  $\rho|_T \doteq \rho \cap T \times T$  be the restriction of  $\rho$  to  $T$ . A linear order  $\tau \subseteq T \times T$  is a *linear extension of  $\rho$  on  $T$*  if  $\tau$  is a linear extension of  $\rho|_T$ . We define  $\text{Lin}(\rho, T)$  to be the set of linear extensions of  $\rho$  on  $T$ .

**Notation on Graphs.** Given an undirected graph  $G = (V, E)$  and a vertex  $v \in V$ , we let  $N(v) \doteq \{u \mid u \in V, (v, u) \in E\}$  be the neighborhood of  $v$ .

A path decomposition of a graph  $G = (V, E)$  is a sequence  $D = (B_1, B_2, \dots, B_r)$  of subsets of  $V$ , such that the following conditions are satisfied.

- $\bigcup_{1 \leq i \leq r} B_i = V$ .
- For each edge  $(u, v) \in E$ , there is an  $i \in [r]$  such that  $u, v \in B_i$ .
- For each  $i, j, k \in [r]$  with  $i < j < k$ ,  $B_i \cap B_k \subseteq B_j$ .

The *width* of  $D$  is defined as  $w(D) = \max_{i \in [r]} |B_i| - 1$ . The *pathwidth*,  $pw(G)$ , of  $G$  is the minimum width of a path decomposition of  $G$ .

**Partial Orders and Interval Width.** Given a (strict) partial order  $\rho \subseteq V \times V$ , the undirected graph  $G_\rho \doteq (V, E)$  with  $E \doteq \{\{u, v\} \in V \times V \mid u \neq v, (u, v) \notin \rho, (v, u) \notin \rho\}$  is the *cocomparability graph* of  $\rho$ . An *interval order* is a strict partial order  $\iota \subseteq V \times V$  whose elements  $v \in V$  are represented by half-open intervals  $I_v = [l_v, r_v)$  on the real line with  $(u, v) \in \iota \iff r_u \leq l_v$ .  $\{I_v \mid v \in V\}$  is called an *interval representation* of  $\iota$ . The cocomparability graph  $G_\iota$  is the intersection graph of  $\{I_v \mid v \in V\}$  and is hence an interval graph. It is known [29] that interval graphs are exactly the cocomparability graphs that do not contain an induced cycle of length four. The *interval width* of a partial order  $\rho \subseteq V \times V$  is defined as  $iw(\rho) \doteq \min\{w(\iota) \mid \iota \text{ interval order, } \iota \subseteq \rho\}$ , where  $w(\iota)$  is the maximum size of an antichain of  $\iota$ . By Theorem 2.1 from [31],  $pw(G_\rho) = iw(\rho) - 1$ . Conversely, for any graph  $G = (V, E)$ ,  $pw(G) = \min\{\omega(H) \mid H \text{ is an interval graph, } V(H) = V, E(H) \supseteq E\} - 1$ , where  $\omega(H)$  is the size of the largest clique in  $H$ . For more information on interval orders, we refer to textbooks and survey articles such as [26, 54].

### 3 Completion of an Ordering (CO)

Below, we formally define the COMPLETION OF AN ORDERING problem, generalizing POSITIVE COMPLETION OF AN ORDERING (PCO) introduced in [16, Sec. 8] and [23, Sec. 6.4].

**Problem name:** COMPLETION OF AN ORDERING (CO)

**Given:** A partial order  $\rho \subseteq V \times V$ , a cost function  $\mathbf{c} : V \times V \rightarrow \mathbb{N}$ , and  $k \in \mathbb{N}$ .

**Output:** Is there a linear order  $\tau \supseteq \rho$  with  $\mathbf{c}(\tau \setminus \rho) = \sum_{(x,y) \in \tau \setminus \rho} \mathbf{c}(x, y) \leq k$ ?

In the PCO problem, the cost function needs to satisfy the following condition: for all pairs  $(x, y) \in V \times V$  such that  $x$  and  $y$  are incomparable in  $\rho$ ,  $\mathfrak{c}(x, y) > 0$ .

Let us shortly discuss the *cost parameter*  $k$ : By the result of Dujmovic, Fernau and Kaufmann [16] (for details, see [23]), PCO can be solved in time  $\mathcal{O}^*(1.52^k)$  and admits a linear-size kernel. The best known algorithm for PCO, whose running time is  $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k} \log(k))})$ , was obtained in [25] by relating PCO to the FEEDBACK ARC SET PROBLEM IN TOURNAMENTS, or FAST for short, that allows for subexponential algorithms due to [1]. Here, we are presenting an algorithm for a variation of this problem that runs in time  $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$  and is relatively straightforward to implement. We also present a branching algorithm that runs in time  $\mathcal{O}^*(1.42^k)$ , improving on [23]. Our algorithms are based on the interval width of  $\rho$ .

### 3.1 CO, parameterized by pathwidth

Let  $G = (V, E)$  be a graph,  $\rho \subseteq V \times V$  be a (strict) partial order on the vertices of  $G$  and  $D = (B_1, \dots, B_r)$  be a path decomposition of  $G$ . We call  $D$  *consistent* with  $\rho$  if there is no pair of vertices  $(x, y) \in \rho$  with  $\max\{i \in [r] \mid y \in B_i\} < \min\{i \in [r] \mid x \in B_i\}$ . Thus, if  $x$  is smaller than  $y$  in  $\rho$ , then  $y$  cannot be forgotten in  $D$  before  $x$  is introduced in  $D$ . The *consistent pathwidth*,  $\text{cpw}(G, \rho)$ , of  $G$  is the minimum width of a path decomposition of  $G$  consistent with  $\rho$ . We will be interested in particular in the consistent pathwidth  $\text{cpw}(G_\rho, \rho)$ .

► **Theorem 1.** *Given a partial order  $\rho$  over a set  $V$ , a cost function  $\mathfrak{c} : V \times V \rightarrow \mathbb{N}$  and a width- $w$  path decomposition  $D$  of the cocomparability graph  $G_\rho$  that is consistent with  $\rho$ , one can solve an instance  $(\rho, \mathfrak{c}, k)$  of the CO problem in time  $\mathcal{O}(|V| \cdot w \cdot 2^w \cdot \log(k) + |V|^2 \cdot \log(k))$ .*

The remainder of this subsection is dedicated to the proof of Theorem 1.

Let us explain why our pathwidth measure can be seen as a *distance to triviality* parameterization in the context of CO. A trivial instance of CO is a linear order, as it has cost zero. Then, the cocomparability graph is an independent set and has consistent pathwidth  $0^2$ . In the opposite case, if the input partial order is empty, then the cocomparability graph is a clique and has consistent pathwidth  $|V| - 1$ . It is also worth noticing that it is NP-hard to determine the pathwidth of a cocomparability graph, together with an optimal path decomposition, as observed in [31].

**Notation on Path Decompositions.** Let  $D = (B_1, B_2, \dots, B_r)$  be a path decomposition of a graph  $G$ . We say that  $[r]$  is the set of positions of  $D$  and that  $r$  is the length of  $D$ . For each position  $i$ , we say that  $B_i$  is the  $i$ -th bag of  $D$ . For each  $i \in [r]$ ,  $i > 1$ , we say that  $B_i$  is an *introduce bag* if  $B_i = B_{i-1} \cup \{v\}$  and that  $B_i$  is a *forget bag* if  $B_i = B_{i-1} \setminus \{v\}$ . We say that the path decomposition  $D = (B_1, B_2, \dots, B_r)$  is *nice* if for each  $i \in [r]$ ,  $B_i$  is either an introduce bag or a forget bag and  $|B_1| = 1$  and  $B_r = \emptyset$ . It can be shown that, given any path decomposition  $D = (B_1, B_2, \dots, B_r)$  of width  $w$  of a graph  $G$ , one can construct in time  $\mathcal{O}(r \cdot w(D))$  a nice path decomposition of  $G$  of width at most  $w$ . In a nice path decomposition, for every vertex of  $V$ , there is a bag that introduces it and a bag that forgets it, so the length of a nice path decomposition is  $2 \cdot |V|$ . For each position  $i \in [r]$ , we let  $L_i = \bigcup_{1 \leq j \leq i-1} B_j \setminus B_i$  be the set of vertices that have been forgotten (lost) up to position  $i$ .

► **Lemma 2.** *Let  $\iota$  be an interval order over  $V$  and  $\{I_v \mid v \in V\}$  be an interval representation of  $\iota$ . One can derive a minimum width path decomposition of  $G_\iota$  consistent with  $\iota$  from  $\{I_v \mid v \in V\}$  of width  $w(\iota) - 1$  in time  $\mathcal{O}(w(\iota) \cdot |V|)$ .*

<sup>2</sup> In Lemma 5, we show that consistent pathwidth is equal to pathwidth.

**Proof.** For each element  $v$  in  $V$ , we let  $l_v$  and  $r_v$  be the left and right endpoints of  $I_v$ . For every point  $x$  on the real line  $\mathbb{R}$  that corresponds to an endpoint of one or more intervals, we associate a bag  $B_x = \{v \mid x \in I_v\}$ . Then, we order the bags following the order of  $l_v$  and  $r_v$  on the real line. For each element  $v \in V$ ,  $v \in B_{l_v}$ . Given three bags  $B_x, B_y, B_z$  such that  $x \leq y \leq z$ , we have that  $B_x \cap B_z = \{v \mid x \in I_v\} \cap \{v \mid z \in I_v\} = \{v \mid l_v \leq x \leq z < r_v\} \subseteq \{v \mid l_v \leq y < r_v\} = B_y$ . For each edge  $(u, v) \in E(G_\iota)$ ,  $I_u$  and  $I_v$  intersect, therefore, we have either  $l_v \in I_u$  or  $l_v \in I_u$ . If  $l_v \in I_u$  then  $u, v \in B_{l_v}$ , similarly if  $l_u \in I_v$  then  $u, v \in B_{l_u}$ . So this construction builds a path decomposition. We call this path decomposition  $D$ . Now, we will show that  $D$  is consistent with  $\iota$ . More precisely, we will show that for each pair  $(u, v) \in \iota$ ,  $\max\{x \in \mathbb{R} \mid u \in B_x\} < \min\{x \in \mathbb{R} \mid v \in B_x\}$ . For each  $(u, v) \in \iota$ , we have  $l_u < r_u \leq l_v$ ,  $\max\{x \in \mathbb{R} \mid u \in B_x\} < r_u \leq l_v \leq \min\{x \in \mathbb{R} \mid v \in B_x\}$ . Therefore,  $D$  is consistent with  $\iota$ . Note that each bag is a clique, therefore, this is a path decomposition of minimum width. A clique in  $G_\iota$  is an antichain of  $\iota$  and each antichain of  $\iota$  forms a clique in  $G_\iota$ . Therefore, we have that  $D$  has width  $w(\iota) - 1$ . ◀

We will refer to this decomposition as the path decomposition *derived* from the interval order  $\iota$ .

► **Lemma 3.** *Let  $G = (V, E)$  be a graph. Given a partial order  $\rho$  on  $V$  and a path decomposition  $D$  of  $G$  of width  $w$  and length  $r$  that is consistent with  $\rho$ , one can construct in time  $\mathcal{O}(w \cdot r)$  a nice path decomposition of width  $w$  that is consistent with  $\rho$ .*

Given a path decomposition  $D$ , one can get a nice path decomposition by introducing before each bag  $B$  several new bags that will forget one by one each vertex forgotten by  $B$  and introduce each new vertex in  $B$  one by one. If  $D$  is consistent with  $\rho$ , then the new path decomposition is also consistent with  $\rho$ . For the cocomparability graph, we can further show:

► **Lemma 4.** *Let  $\rho$  be a partial order on a set  $V$ ,  $G_\rho$  be the cocomparability graph of  $\rho$  and  $D$  be a path decomposition of  $G_\rho$  consistent with  $\rho$ , then  $D$  is consistent with any extension of  $\rho$ .*

► **Lemma 5.** *Let  $G_\rho = (V, E)$  be the cocomparability graph of a partial order  $\rho \subseteq V \times V$ . Then  $pw(G_\rho) = cpw(G_\rho, \rho)$ .*

**Proof.** By definition we have  $pw(G_\rho) \leq cpw(G_\rho, \rho)$ . We will show that  $cpw(G_\rho, \rho) \leq iw(\rho) - 1$  and use the fact that  $pw(G_\rho) = iw(\rho) - 1$  (Theorem 2.1 from [31]). By definition of  $iw(\rho)$ , we can find an interval order  $\iota$  such that  $iw(\rho) = w(\iota)$  and  $\iota \subseteq \rho$ . Let  $\{I_v \mid v \in V\}$  be an interval representation of  $\iota$ . Then  $G_\iota$  is the intersection graph of  $\{I_v \mid v \in V\}$ . Then by Lemma 2, the path decomposition  $D$  of  $G_\iota$  derived from  $\{I_v \mid v \in V\}$  is consistent with  $\iota$  and has width  $w(\iota) - 1$ . From Lemma 4, we know that  $D$  is also consistent with the extension  $\rho$  of  $\iota$ . We conclude  $cpw(G_\rho, \rho) \leq cpw(G_\iota, \iota) = w(\iota) - 1 = iw(\rho) - 1 = pw(G_\rho)$ . ◀

**Dynamic Programming Algorithm.** Let  $\rho \subseteq V \times V$  be a partial order over a set  $V$ ,  $c : V \times V \rightarrow \mathbb{N}$  be a cost function and  $S$  and  $T$  be two subsets of  $V$  such that for each pair  $(s, t) \in S \times T$ ,  $(t, s) \notin \rho$ . We define  $\mathfrak{c}(S, T) = \sum_{(s, t) \in (S \times T) \setminus \rho} c(s, t)$ , this is the cost of having elements of  $S$  before elements of  $T$ . For every linear extension  $\tau$  of  $\rho$  on  $T$ , we let  $\mathfrak{c}(\tau) = \sum_{(a, b) \in \tau \setminus \rho|_T} c(a, b)$  be the cost of  $\tau$ . We define  $\text{opt}(T) = \min\{\mathfrak{c}(\tau) \mid \tau \in \text{Lin}(\rho, T)\}$ . Our goal is to find  $\text{opt}(V)$ .

Let  $D$  be a path decomposition of width  $w$  of the graph  $G_\rho$  consistent with  $\rho$ . By Lemma 3, we can assume without loss of generality that  $D$  is nice.

## 9:6 Width Notions for Ordering-Related Problems

For each position  $1 \leq i \leq 2 \cdot |V|$  in the path decomposition, we compute and store  $\mathbf{c}(L_i, \{v\})$  for every vertex  $v \in B_i$  such that for each  $u \in L_i$   $(v, u) \notin \rho$  in table  $T_i^c$  and  $\text{opt}(L_i \cup T)$  for each  $T \subseteq B_i$  in table  $T_i^{\text{opt}}$ . For every vertex  $v \in B_i$  such that for each  $u \in L_i$   $(v, u) \notin \rho$ ,  $\mathbf{c}(L_i, \{v\})$  is the cost of having  $v$  after the vertices forgotten at position  $i$  if this is compatible with  $\rho$  and for each  $T \subseteq B_i$ ,  $\text{opt}(L_i \cup T)$  is the minimum cost of a linear extension on  $L_i \cup T$ . We have  $L_{2 \cdot |V|} \cup B_{2 \cdot |V|} = V$ . So, to find the solution, it is enough to inductively construct these two tables. The induction basis is trivial:  $L_1 = \emptyset$  and  $|B_1| = 1$ , so that  $\mathbf{c}(L_1, \{v\}) = 0$  for every vertex  $v \in B_1$  in table  $T_1^c$  and  $\text{opt}(L_i \cup T) = 0$  for both  $T = \emptyset$  and  $T = B_1$  in table  $T_1^{\text{opt}}$ . The following two lemmas explain the induction step of the algorithm.

► **Lemma 6.** *Let  $i \in [2, \dots, 2 \cdot |V|]$ . Given a table  $T_{i-1}^c$  that lists the values of  $\mathbf{c}(L_{i-1}, \{v\})$  for every  $v \in B_{i-1}$ , one can compute  $\mathbf{c}(L_i, \{v\})$  for every  $v \in B_i$  in time  $w \cdot \log(k)$  in order to build the table  $T_i^c$ .*

► **Lemma 7.** *Let  $i \in [2, \dots, 2 \cdot |V|]$ . Given a table  $T_i^c$  that lists the values of  $\mathbf{c}(L_i, \{v\})$  for every  $v \in B_i$  such that for each  $u \in L_i$   $(v, u) \notin \rho$  and a table  $T_{i-1}^{\text{opt}}$  that lists the values of  $\text{opt}(L_{i-1} \cup T)$  for every  $T \subseteq B_{i-1}$ , one can compute in  $\mathcal{O}(w \cdot 2^w \cdot \log(k))$  time the value of  $\text{opt}(L_i \cup T)$  for all  $T \subseteq B_i$  in order to build the table  $T_i^{\text{opt}}$ .*

**Proof.** The cost can be arbitrarily large, therefore, the addition of two costs is done in time  $\mathcal{O}(\log(k))$ . First, we compute  $\mathbf{c}(T, \{v'\})$  for  $v' \in B_i$  and for  $T \subseteq B_i \setminus \{v'\}$ , and store the values in an auxiliary table  $T^{\text{aux}}$ . This computation can be done in  $\mathcal{O}(w \cdot 2^w \cdot \log(k))$  time. Now there are two cases:

- If  $B_i$  forgets a vertex  $v$ , then  $L_i = L_{i-1} \cup \{v\}$ ; for each subset  $T \subseteq B_i$ ,  $\text{opt}(L_i \cup T) = \text{opt}(L_{i-1} \cup T \cup \{v\})$  and this value is in the table  $T_{i-1}^{\text{opt}}$ , as  $T \cup \{v\} \subseteq B_{i-1}$ .
- If  $B_i$  introduces a vertex  $v$ , then  $L_i = L_{i-1}$  and  $B_i = B_{i-1} \cup \{v\}$ . Given a subset  $T$  of  $B_i$ , if  $v \notin T$ , then  $\text{opt}(L_i \cup T)$  is already in the table  $T_{i-1}^{\text{opt}}$ . Suppose  $v \in T$ . For all  $u \in L_i$ , there is no edge between  $u$  and  $v$  in  $G_\rho$ , and as  $D$  is consistent with  $\rho$ , we have  $(u, v) \in \rho$ . So in any linear extension of  $\rho$  on  $L_i \cup T$ , the maximum element is a maximal element of  $T$  (with respect to  $\rho$ ). Then we have, by testing all possible maximum elements  $v'$ :

$$\begin{aligned} \text{opt}(L_i \cup T) &= \min_{v' \in \max_\rho(T)} \{ \text{opt}(L_i \cup T \setminus \{v'\}) + \mathbf{c}(L_i \cup T \setminus \{v'\}, \{v'\}) \} \\ &= \min_{v' \in \max_\rho(T)} \{ \text{opt}(L_i \cup T \setminus \{v'\}) + \mathbf{c}(L_i, \{v'\}) + \mathbf{c}(T \setminus \{v'\}, \{v'\}) \} \end{aligned}$$

where  $\max_\rho(T) = \{v \in T \mid \forall u \in T, (v, u) \notin \rho\}$  is the set of maximal elements of  $T$  with respect to  $\rho$ . The second and third terms are in the tables  $T_i^c$  and  $T^{\text{aux}}$ , respectively. If  $v' = v$ , then the first term can be looked up in table  $T_{i-1}^{\text{opt}}$ . By walking through all  $T \subseteq B_i$  with increasing cardinality (recall that always  $v \in T$ ), we can inductively compute  $\text{opt}(L_i \cup T)$ , as this provides the first term. As inductive basis, consider  $T = \{v\}$ , in which case  $\text{opt}(L_i \cup T) = \text{opt}(L_i \cup \{v\}) = \text{opt}(L_i) + \mathbf{c}(L_i, \{v\})$ . The first term is already in the table  $T_{i-1}^{\text{opt}}$ . The computation of  $T_i^{\text{opt}}$  can be done in time  $\mathcal{O}(w \cdot 2^w \cdot \log(k))$ .

This explains how to build the table  $T_i^{\text{opt}}$ . ◀

Since  $L_{2 \cdot |V|} \cup B_{2 \cdot |V|} = V$ , the dynamic programming algorithm can provide an optimal solution and runs in time  $\mathcal{O}(|V| \cdot w \cdot 2^w \cdot \log(k))$ . The size of the cost function given as input is  $|V|^2 \cdot \log(k)$ . Reading the cost function gives the second part of the running time. This proves Theorem 1.

### 3.2 Further Algorithmic Consequences

The relation between variants of FAST and CO range in both directions. One direction (solving PCO with the help of FAST) was exploited in [25]. We are now explaining a reverse reduction. The CONSTRAINED FAST problem [9, 57] is defined as follows: The arc set of a given tournament graph is split into fixed arcs  $A_{\text{fix}}$  and free arcs  $A_{\text{free}}$ . The task is to remove at most  $k$  free arcs such that the resulting graph becomes acyclic. We know that every arc in  $A_{\text{free}}$  that contradicts the transitivity of  $A_{\text{fix}}$  needs to be removed. Therefore, we assume that  $A_{\text{fix}}$  gives a transitive relation on the set of vertices and that it is acyclic, so that it defines a partial order  $\rho$  on the vertex set  $V$ . By defining the following cost function, we can solve CONSTRAINED FAST with any CO algorithm: For arcs  $(x, y) \in A_{\text{free}}$ , we set  $\mathbf{c}(x, y) = 0$ . For arcs  $(x, y)$  such that  $(y, x) \in A_{\text{free}}$ , we set  $\mathbf{c}(x, y) = 1$ . By the tournament condition, for each edge  $\{x, y\}$  of  $G_\rho$ ,  $\mathbf{c}(x, y) \in \{0, 1\}$  and  $\mathbf{c}(y, x) \in \{0, 1\}$  are defined, with  $\mathbf{c}(x, y) + \mathbf{c}(y, x) = 1$ . As FAST is a well-known NP-complete problem, this also shows NP-completeness for CO (even with costs 0, 1 only) and similarly, we obtain NP-completeness for PCO, even with costs from the set  $[2] = \{1, 2\}$ .

**Completing an interval ordering is easier.** Consider the following restriction of PCO:

**Problem name:** POSITIVE COMPLETION OF AN INTERVAL ORDERING (PCIO)  
**Given:** An interval order  $\iota \subseteq V \times V$  over a set  $V$ , a cost function  $\mathbf{c} : V \times V \rightarrow \mathbb{N}$  satisfying  $\forall x, y \in V : ((x, y) \notin \iota \wedge (y, x) \notin \iota) \implies \mathbf{c}(x, y) > 0$ , and an integer  $k \in \mathbb{N}$ .  
**Output:** Is there a linear order  $\tau \supseteq \iota$  with  $\mathbf{c}(\tau \setminus \iota) \leq k$ ?

This variation has two more restrictions compared to CO: the cost between two incomparable elements must not be zero and the partial order is an interval order. These two restrictions allow us to get better bounds for our dynamic programming algorithm.

► **Theorem 8.** *An instance  $(\iota, \mathbf{c}, k)$  of PCIO is solvable in time  $\mathcal{O}(k \cdot 2^{\sqrt{2k}} \cdot \log(k) + |V|^2 \cdot \log(k))$ .*

The following is an outline of our algorithm, called DP-PCIO.

1. Construct  $G_\iota$ , if  $G_\iota$  has more than  $k$  edges then stop with “NO”. This can be done in time  $|V|^2$ . This is justified, because  $\mathbf{c}(x, y) > 0$  for each incomparable pair  $\{x, y\}$ .
2. Construct a nice path decomposition  $D$  consistent with  $\iota$ . If the width of  $D$  is more than  $\sqrt{2k}$ , then stop with “NO”, as a large clique was detected.
3. Compute  $\text{opt}(V)$  by a dynamic programming algorithm based on the path decomposition  $D$ . If the current optimum solution is bigger than  $k$ , then stop with “NO”. If the computation is successful and  $\text{opt}(V) \leq k$  then answer “YES”. Otherwise answer “NO”.

We will prove several lemmas to show Theorem 8. To apply our dynamic programming algorithm, we need consistency.

► **Lemma 9.** *Given an interval order  $\iota$ , one can construct in linear time a path decomposition of  $G_\iota$  consistent with  $\iota$  of minimum width.*

Let  $D = (B_1, \dots, B_{2|V|})$  be the nice path decomposition consistent with  $\iota$  we got by applying Lemma 3 on the path decomposition of Lemma 9. Clearly, each bag in  $D$  is a clique.

► **Lemma 10.** *Assume that  $G_\iota$  has at most  $k$  edges. Let  $H = \lceil \sqrt{2k} \rceil + 1$  and, for  $2 \leq h \leq H$ , let  $c_h \doteq |\{i : |B_i| = h\}|$ . Then we have  $c_h \leq k/(h-1) - h/2 + 1$ .*

Finally, we show how our considerations also help to improve the running time of a simple branching algorithm. The algorithm works as follows: it picks an edge in the cocomparability graph and considers orienting it both ways. As long as there are *profitable edges* that cause at least a cost of two in each branch, we keep on branching. Costs can also be implicitly caused, as we modify the partial order  $\rho$  and hence transitivity must be maintained. We can use Theorem 8 when there are no more profitable edges because of the following lemma.

► **Lemma 11.** *After exhaustively branching at all profitable edges,  $G_\rho$  is an interval graph.*

This is the key to the following improvement on the branching algorithm described in [23]. Notice that in practice, branching algorithms tend to be faster at least for small parameter values, due to the smaller constants in the basis of the (sub-)exponential functions that upper-bound the running times.

► **Theorem 12.** *PCO can be solved in time  $\mathcal{O}^*(\sqrt{2}^k)$  by a branching algorithm.*

#### 4 One-Sided Crossing Minimization (OSCM)

Given a bipartite graph  $G$  with bipartition  $(V_1, V_2)$ , a two-layer drawing of  $G$  is a drawing such that vertices of  $V_1$  and  $V_2$  are placed on two parallel lines and edges are represented as straight lines between the vertices. A two-layer drawing can be specified by two linear orders  $\tau_1$  of  $V_1$  and  $\tau_2$  of  $V_2$ . A crossing in a two-layer drawing is a pair of edges that intersect each other in a point that is not a vertex. The number of crossings is defined by the order of  $V_1$  and  $V_2$  on the lines. The ONE-SIDED CROSSING MINIMIZATION problem consists in placing vertices of one part  $V_2$  of the bipartite graph, given an ordering of the other part  $V_1$ , that minimizes the number of crossings. This problem is a key sub-problem for drawing hierarchical graphs [3, 4, 32, 45] or producing row-based VLSI layouts [50, 52].

**Problem name:** ONE-SIDED CROSSING MINIMIZATION (OSCM)

**Given:** A bipartite graph  $G = (V_1, V_2, E)$ , a linear order  $\tau_1$  on  $V_1$  and  $k \in \mathbb{N}$

**Output:** Is there a linear order  $\tau_2$  on  $V_2$  such that, in the two-layer drawing specified by  $(\tau_1, \tau_2)$ , at most  $k$  edge crossings incur?

The problem is known to be NP-complete [21] even in sparse graphs [44] and FPT in the number of edge crossings  $k$  [17, 18, 25], including sub-exponential algorithms. The two-sided variant of the problem (where the permutation of both sides is variable) is also FPT in the number of crossings [37]. OSCM is a cornerstone of algorithms dealing with the so-called *Sugiyama approach* to hierarchical graph drawing, see [32, 53].

Now, we show that OSCM can be reduced into PCIO, starting with a simple remark.

► **Remark 13.** Isolated vertices in  $V_2$  can be placed anywhere in an optimal ordering of  $V_2$ . From here, we assume that  $V_2$  does not contain any isolated vertices. Similar to [23, 25, 38], we can model OSCM instances as PCIO instances.

► **Lemma 14.** *Given an instance  $(G, \tau_1, k)$  of OSCM, one can construct in polynomial time an equivalent instance  $(\iota, c, k)$  of PCIO.*

As PCIO has not been formally studied in the literature, let us draw an important consequence from the previous lemma (also see the discussion in the beginning of Section 3.2).

► **Corollary 15.** *POSITIVE COMPLETION OF AN INTERVAL ORDERING is NP-complete, even when restricted to instances  $(\iota, c, k)$  where the arc weights are within the set  $\{1, 2, \dots, 16\}$ .*

► Remark 16. We can use Theorem 8 to immediately deduce an algorithm for OSCM matching the running time  $\mathcal{O}^*(2^{\sqrt{2k}})$  of the best published algorithm for OSCM [38]. We could also use the PCO-kernelization as a kernelization procedure for OSCM.

► Remark 17. Çakiroglu *et al.* [11] studied the variation where edges (if existing) have positive weights, and the cost of an edge crossing is obtained by the product of the weights of the crossing edges. This modification (with applications in automatic graph drawing) can also be modeled by PCIO, so that we inherit an  $\mathcal{O}^*(2^{\sqrt{2k}})$  algorithm for the standard parameter  $k$ .

## 5 The Kemeny Rank Aggregation Problem

Preference lists are extensively used in social science surveys and voting systems to capture information about choice. Kemeny [36] discussed the problem to combine several preference lists into one, called its aggregation. This approach aims at minimizing the total disagreement (formalized below) between the several input rankings and their aggregation. The idea itself has not only applications in (the theory of) elections in the context of social sciences, say, on a committee, but has also been suggested as a means of designing meta-search engines [19]. It has been also shown by Young and Levenglick [56] that the aggregation method proposed by Kemeny is the only one satisfying a number of natural requirements on such aggregations.

More formally, in KEMENY RANK AGGREGATION we are given a set  $\Pi$  of rankings (also called *votes*) over a set of alternatives  $C$  (also called *candidates*), and a positive integer  $k$ , and are asked for a ranking  $\pi$  of  $C$ , such that the sum of the *Kendall-Tau distances* (or, KT-distances for short) of  $\pi$  from all the votes, called its *Kemeny score*, is at most  $k$ . The ranking  $\pi$  that gives the smallest Kemeny score is called a *Kemeny consensus*. The KT-distance between two rankings  $\pi_1$  and  $\pi_2$  is the number of pairs of candidates that are ordered differently in the two rankings and is denoted by  $\text{KT-dist}(\pi_1, \pi_2)$ . Hence, if  $\pi_1, \pi_2 : [|C|] \rightarrow C$ ,  $\text{KT-dist}(\pi_1, \pi_2) = |\{(c, c') \in C \times C \mid c <_{\pi_1} c' \wedge c' <_{\pi_2} c\}|$ . Observe that the Kendall-Tau distance can be seen as the “bubble sort” distance.

**Problem name:** KEMENY RANK AGGREGATION (KRA)

**Given:** A list of votes  $\Pi$  over a set of candidates  $C$ , a non-negative integer  $k$

**Output:** Is there a ranking  $\pi$  on  $C$  such that the sum of the KT-distances of  $\pi$  from all the votes is at most  $k$ .

Hence, given rankings  $\pi_1, \dots, \pi_m$  of  $C$  and a non-negative integer  $k$ , the question is if there exists a ranking  $\pi : [|C|] \rightarrow C$  such that  $\sum_{i=1}^m \text{KT-dist}(\pi, \pi_i) \leq k$ . The problem KEMENY RANK AGGREGATION is known to be NP-complete [2], even if only four votes are input [19].<sup>3</sup> Simjour [51] obtained an algorithm for the problem that runs in time  $\mathcal{O}^*(1.403^k)$ . There are also sub-exponential algorithms for KEMENY RANK AGGREGATION under this parameterization: Karpinski and Schudy [35] obtained an algorithm for KEMENY RANK AGGREGATION that runs in  $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k})})$  time, while the algorithm of Fernau *et al.* [24, 25], based on a different methodology, runs in  $\mathcal{O}^*(k^{\mathcal{O}(\sqrt{k})})$  time. Both algorithms hide some constant factor in the  $\mathcal{O}$ -notation in the exponent that is not that clear from the expositions. Our considerations are also valid for *weighted Kemeny score*, a modification suggested in [5] that assigns positive weights to the voters. We can add some comment on conditional lower bounds of this problem by bringing together facts from different parts of the literature.

<sup>3</sup> The proof of this fact is not contained in the conference paper [19] but only appears in Appendix B of [http://www.wisdom.weizmann.ac.il/~naor/PAPERS/rank\\_www10.html](http://www.wisdom.weizmann.ac.il/~naor/PAPERS/rank_www10.html).

► **Theorem 18.** *KRA on instances with only  $m = 4$  votes on some candidate set  $C$  and some integer  $k$  bounding the sum of the Kendall-Tau distances to a solution cannot be solved neither in time  $\mathcal{O}^*(2^{o(|C|)})$  nor in time  $\mathcal{O}^*(2^{o(\sqrt{k})})$ , unless ETH fails.*

### 5.1 Reduction from KRA to PCO

Now we will show that KRA can be encoded into PCO. Let  $(\Pi, C)$  be an instance of KEMENY RANK AGGREGATION with  $m$  votes  $\Pi = (\pi_1, \dots, \pi_m)$  over  $n$  candidates  $C$ . From this instance, we construct an equivalent instance of the PCO problem  $\rho \subseteq V \times V$ ,  $\mathfrak{c}$  with base set  $V = C$ . For every pair of candidates  $c_1$  and  $c_2$ , we define the *cost* of  $(c_1, c_2)$ ,  $\mathfrak{c}(c_1, c_2)$ , as the number of votes that do not order  $c_1$  before  $c_2$ . More formally,  $\mathfrak{c}(c_1, c_2) = |\{i \in [m] \mid c_2 <_{\pi_i} c_1\}|$ .

► **Lemma 19.** *Given two candidates  $c_1$  and  $c_2$ , if for every vote  $\pi_i \in \Pi$ , we have  $c_1 <_{\pi_i} c_2$  then for every Kemeny consensus  $\pi$ ,  $c_1 <_{\pi} c_2$ .*

Using different terminology, a proof of this lemma can be found in [43, Théorème 3]. Now, we define the partial order  $\rho$  as follows:  $(c_1, c_2) \in \rho$  if and only if  $\mathfrak{c}(c_1, c_2) = 0$ . Hence,  $<_{\rho} = \bigcap_{i=1}^m <_{\pi_i}$  is the *unanimity order* [12]. By Lemma 19, a vote  $\pi$ , which is a linear order of the candidates, is a Kemeny consensus iff  $\pi$  is a linear extension of  $\rho$  of minimum cost with Kemeny score<sup>4</sup>

$$\sum_{i=1}^m \text{KT-dist}(\pi, \pi_i) = \sum_{i=1}^m \sum_{j=1}^n \sum_{k=1}^n [c_j <_{\pi_i} c_k \wedge c_k <_{\pi} c_j] = \sum_{j=1}^n \sum_{k=1}^n \mathfrak{c}(c_k, c_j) [c_k <_{\pi} c_j] \quad (1)$$

is equal to the cost of the linear extension given by  $\pi$  according to its definition.

These considerations prove that we can translate our algorithmic results for PCO to KRA.

► **Remark 20.** Our reduction works even if votes are reflexive and antisymmetric relations instead of linear orders. In this case, the cost between  $c_1$  and  $c_2$  is defined as follows:  $\mathfrak{c}(c_1, c_2) = |\{i \in [m] \mid c_1 \not<_{\pi_i} c_2\}|$ .

### 5.2 Pathwidth in Kemeny Rank Aggregation

Now we will discuss the meaning of the pathwidth measure from the PCO problem applied to KRA. For KRA, several measures have been studied in the context of parameterized complexity, Betzler et al. [5] introduced the notion of *maximum range of candidate positions*. For an election  $(\Pi, C)$ , the *range*  $r(c)$  of a candidate  $c$  is defined as  $r(c) \doteq \max_{i,j \in [m]} |\pi_i^{-1}(c) - \pi_j^{-1}(c)| + 1$ . If  $\Pi(c) \doteq \{i \in [|C|] : \exists \pi \in \Pi : \pi(i) = c\}$  denotes the set of positions candidate  $c$  received in election  $(\Pi, C)$ , then  $r(c) = \max \Pi(c) - \min \Pi(c) + 1$ . The *maximum range*  $r_{\max}$  of an election is given by  $r_{\max} \doteq \max_{c \in C} r(c)$ . Betzler et al. [5] proved that KRA can be solved in time  $\mathcal{O}(32^{r_{\max}} \cdot (r_{\max}^2 \cdot |C| + r_{\max} \cdot |C|^2 \log |C| \cdot m) + m^2 \cdot |C| \log |C|) = \mathcal{O}^*(2^{5r_{\max}})$ .

► **Lemma 21.** *Given an election  $(\Pi, C)$ , let  $w$  be the consistent pathwidth associated to the election and  $r_{\max}$  be the maximum range of the election. We have  $w \leq 2 \cdot r_{\max} - 2$ .*

**Proof.** Let  $\rho$  be the partial order defined by the election. To prove this statement, we will construct an interval order  $\iota$  such that  $\iota \subseteq \rho$ , and  $w(\iota) \leq 2 \cdot r_{\max}$ . To each candidate  $c \in C$ , we associate the interval  $I_c \doteq [\min \Pi(c) - 1, \max \Pi(c))$ . (We subtract one from the left border to avoid empty intervals.) We let  $\iota$  be the interval order associated with the interval

<sup>4</sup> Recall the bracket notation: if  $p$  is a logical proposition, then  $[p]$  yields 1 if  $p$  is true and else,  $[p]$  yields 0.

representation  $\{I_c \mid c \in C\}$ . By Lemma 19, we have that  $\rho$  is an extension of the interval order  $\iota$ . Each interval  $I_c$  has length at most  $r_{\max}$ . Thus, there are at most  $2 \cdot r_{\max} - 2$  intervals that intersect at one point in the interval representation. Hence,  $w(\iota) \leq 2 \cdot r_{\max} - 2$ . ◀

Hence, Theorem 1 yields the following noticeable improvement to the mentioned result of [5]:

► **Corollary 22.** *KRA can be solved in time  $\mathcal{O}(|C| \cdot r_{\max} \cdot 2^{2r_{\max}} + m \cdot |C|^2) = \mathcal{O}^*(2^{2r_{\max}})$ .*

## 6 Grouping by Swapping (GbS)

This problem asks whether a given string can be transformed by at most  $k$  interchanges of neighboring letters into a block format where all occurrences of each letter are adjacent to form one single block each. It is on the famous list of NP-complete problems in [28]. Further algorithmic aspects are discussed in [15, 55]. We show that GbS can be reduced to OSCM in a parameter-preserving way and hence inherits FPT-results shown above. We first discuss the problem GbS itself and then continue with the reductions.

**Problem name:** GROUPING BY SWAPPING (GBS)

**Given:** A finite alphabet  $\Sigma$ , a string  $w \in \Sigma^*$ , and  $k \in \mathbb{N}$ .

**Output:** Is there a sequence of at most  $k$  adjacent swaps such that  $w$  is transformed into a string  $w'$  where all occurrences of each symbol are in single blocks?

Let us formalize this problem a bit more. If  $w, w' \in \Sigma^*$  both have length  $n$ , we call  $w'$  a *permutation* of  $w$  if there exists a bijection  $\pi : [n] \rightarrow [n]$  such that, for any  $i \in [n]$ ,  $w'[i] = w[\pi(i)]$ . Slightly abusing notation, we will also write  $w' = \pi(w)$ . Special bijections are adjacent swaps  $\sigma_i : [n] \rightarrow [n]$  (with  $i \in [n-1]$ ) that act as the identity with two exceptions:  $\sigma_i(i) = i + 1$  and  $\sigma_i(i + 1) = i$ . Every bijection  $\pi : [n] \rightarrow [n]$  can be written as a composition of swaps (property (\*)). Hence, given a permutation  $w'$  of  $w$ , we can ask to compute the *swap distance*, written  $sd(w, w')$ , which is the smallest number  $k$  of swaps  $\sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_k}$  such that  $w' = (\sigma_{i_1} \circ \sigma_{i_2} \circ \dots \circ \sigma_{i_k})(w)$ . Observe that  $sd$  can be viewed, for each mapping  $g : \Sigma \rightarrow \mathbb{N}$ , as a metric on the space of all words  $w \in \Sigma^*$  with  $g(a)$  occurrences of  $a$  for each letter  $a \in \Sigma$ . In particular,  $sd(w, w') = sd(w', w)$  for all permutations  $w'$  of  $w$ . Notice that the swap distance can be computed in quadratic time by dynamic programming, as shown in [42] (property (+)).

This picture changes if we add one more degree of freedom. Let us call  $w' \in \Sigma^*$  to be in *block format* if there is a bijection  $f : [|\Sigma|] \rightarrow \Sigma$  such that  $w' \in f(1)^* f(2)^* \dots f(|\Sigma|)^*$ . Alternatively, we can view  $f$  as defining a linear order  $<_f$  on  $\Sigma$ , and then the block format of  $w$  corresponding to  $f$  is the  $<_f$ -lexicographic smallest permutation of  $w$ . GbS now asks, given  $w \in \Sigma^*$  and  $k \geq 0$ , if there is some permutation  $w'$  of  $w$  that is in block format and has swap distance at most  $k$  from  $w$ . As claimed in [28], this variant is NP-complete. Unfortunately, the proof referenced by [28] is hidden in a private communication. We remedy this below by proving that GbS is NP-complete even for strings  $w$  where each letter occurs exactly four times. Let us start with two rather straightforward observations.

► **Lemma 23.** *Any string  $w$  can be grouped into blocks using at most  $|w|^2$  many swaps.*

In fact, any permutation of  $w$  can be obtained by using at most  $|w|^2$  many swaps, as can be seen by bubble-sort. This reasoning also shows (\*), as well as (+), with a little bit of thinking. This can be used to obtain our first (easy) FPT-result, to be improved on later.

► **Lemma 24.** *GbS on strings  $w \in \Sigma^n$  parameterized by  $|\Sigma|$  can be solved in time  $\mathcal{O}^*(|\Sigma|!)$ .*

We are now going to show that computing the swap distance can be done by considering the distance for pairs of letters, summing up the corresponding results. Notice that the formula in Lemma 25 resembles earlier derived summation formulae, as the defining equation for PCO. To make this more precise, let  $\Sigma' \subseteq \Sigma$  and consider the projection  $p_{\Sigma, \Sigma'} : \Sigma \rightarrow \Sigma'$  that maps  $a \mapsto a$  for  $a \in \Sigma'$  and  $a \mapsto \varepsilon$ , the empty word, if  $a \notin \Sigma'$ , as a morphism  $\Sigma^* \rightarrow (\Sigma')^*$ .

► **Lemma 25.** *Let  $w, w' \in \Sigma^*$  such that  $w'$  is a permutation of  $w$ . Let  $w'$  be in block format following the linear order  $\tau$  on  $\Sigma$ .  $sd(w, w') = \sum_{a, b \in \Sigma, a <_{\tau} b} sd(p_{\Sigma, \{a, b\}}(w), p_{\Sigma, \{a, b\}}(w'))$ . Moreover,  $p_{\Sigma, \{a, b\}}(w') = a^{|w|_a} b^{|w|_b}$  if  $a <_{\tau} b$ .*

## 6.1 Discussing NP-completeness

In this subsection, we will prove NP-completeness of GBS even for quite restricted instances by making use of a somewhat similar result for OSCM, based on [44].

► **Theorem 26.** *GBS is NP-complete, even if each letter has exactly 4 occurrences.*

**Proof.** Membership in NP is clear. In order to show NP-hardness, we give a reduction from OSCM which is also NP-complete if each node in  $V_2$  has degree four and each vertex in  $V_1$  has degree one, i.e., if the graph is a forest of 4-stars [44], with all star's centers in  $V_2$ . Let  $G = (V_1, V_2, E)$  be an instance of OSCM with order  $\tau_1$  on  $V_1$  and integer  $k$  such that all vertices in  $V_1$  are of degree one and all vertices in  $V_2$  are of degree four. We set  $\Sigma = V_2 = \{v_1, v_2, \dots, v_n\}$ . Clearly,  $|V_1| = 4n$ . We construct  $w \in \Sigma^{4n}$  (starting from the empty word) by going through the vertices in  $V_1$ , following the order  $\tau_1$ . If the current vertex is adjacent to  $v_i$ , we concatenate  $v_i$  to  $w$ . As the vertices in  $V_1$  are of degree one, this assignment is unambiguous. Following [21], for vertices  $v_i, v_j \in V_2$ , let  $c_{v_i v_j}$  be the number of crossings between edges incident to  $v_i$  and edges incident to  $v_j$  when  $v_i$  is placed left of  $v_j$ . Lemma 3 in [21] states, referring to [20], that for a linear order  $\tau_2$  on  $V_2$ , the number of crossings  $cross(G, \tau_1, \tau_2)$  of the edges between  $V_1$  in order  $\tau_1$  and  $V_2$  in order  $\tau_2$  is  $cross(G, \tau_1, \tau_2) = \sum_{v_i, v_j \in V_2, v_i <_{\tau_2} v_j} c_{v_i v_j}$ . Clearly, for  $v_i, v_j \in V_2$  the number of crossings  $c_{v_i v_j}$  is equal to  $sd(p_{\Sigma, \{v_i, v_j\}}(w), p_{\Sigma, \{v_i, v_j\}}(w_{\tau_2}))$ , where  $w_{\tau_2}$  is the  $\tau_2$ -lexicographic smallest permutation of  $w$ . Combining this observation with Lemma 25, we obtain that for every linear order  $\tau_2$ ,  $sd(w, w_{\tau_2}) = cross(G, \tau_1, \tau_2)$ . ◀

In the following subsection, we will show that, in a sense, the reduction presented in our NP-hardness result for GBS can be reversed. This also shows the following:

► **Remark 27.** GBS is polynomial-time solvable when each letter occurs at most twice.

This also leaves the following case as an **open question**: Can GBS instances be solved in polynomial time if each letter occurs at most thrice? Notice that it is also open whether subcubic OSCM graph instances can be solved in polynomial time. Furthermore, within KRA, it is open if instances with three voters can be solved in polynomial time. Also, Cor. 15 leaves some room for improvement.

## 6.2 Reduction from GbS to OSCM

With the same idea as in the proof of Theorem 26, we can also reduce GBS to OSCM by representing the string  $w$  as the ordered vertex set  $V_1$  and  $\Sigma$  as the vertex set  $V_2$ . More precisely, let  $n$  be the length of  $w$  and interpret  $w$  as a mapping from  $[n]$  into  $\Sigma$ . Moreover, set  $V_1 = [n]$  with the usual linear ordering  $<_{\tau_1} \doteq <$  on  $[n]$ . Let  $V_2 = \Sigma$  and connect  $a \in V_2$  to  $i \in [n]$  iff  $w(i) = a$ . This defines the bipartite graph  $G = (V_1, V_2, E)$  with linear ordering

$\tau_1$  on  $V_1$ . Now, the GBS instance  $(w, k)$  is a YES-instance if and only if the constructed OSCM instance  $(G, \tau_1, k)$  is a YES-instance. As OSCM is solvable in polynomial time if the vertices in  $V_2$  have degree at most two, this implies that GBS is solvable in polynomial time if each letter has at most two appearances, see Remark 27 and the following comments.

Together with the reduction proving Theorem 26, we see that GBS can be viewed as exactly the special case of OSCM where all vertices of  $V_1$  are of degree one, so that the instance becomes a forest of stars with centers in  $V_2$ . We make the algorithmic consequences of this connection explicit, each time giving references to the literature on OSCM.

► **Corollary 28.** *GBS, parameterized by  $k$ , can be solved (in polynomial space) in time  $\mathcal{O}^*(1.4656^k)$  by [17] or (in exponential space) in time  $\mathcal{O}^*(2^{\sqrt{2k}})$  by [38] or Remark 16.*

Fernau *et al.* [25] and Kobayashi and Tamaki [38] also obtained OSCM lower bound results, based on [44], assuming ETH. By the proof of Theorem 26, we can strengthen them:

► **Corollary 29.** *GBS on strings of length  $n$  over alphabet  $\Sigma$ , parameterized by the number  $k$  of swaps, cannot be solved neither in time  $\mathcal{O}^*(2^{o(n)})$  nor in time  $\mathcal{O}^*(2^{o(|\Sigma|)})$  nor in time  $\mathcal{O}^*(2^{o(\sqrt{k})})$ , unless ETH fails, even if each letter has exactly 4 occurrences.*

### 6.3 Pathwidth in Grouping by Swapping

The concept of scope coincidence degree (SCD for short) was introduced in [49] for patterns, which are strings over two disjoint alphabets, where only the alphabet of variables was used to measure the SCD of patterns. We adapt it in the following to strings over a single alphabet.

Given a string  $w \in \Sigma^*$ , and a letter  $a \in \Sigma$ , then the *scope* of  $a$ , denoted  $Scope(a)$  is the set of positions in  $\{1, \dots, |w|\}$  between the minimum position and the maximum position in which  $a$  occurs. For each position  $i$ , we let the incidence set of  $i$  to be  $Inc(i) = \{a \in \Sigma : i \in Scope(a)\}$ . Now the scope coincidence degree is the number of overlapping scopes for all letters. In other words, we have that  $SCD(w) = \max_i |Inc(i)|$ .

Our reduction from GBS to OSCM first turns  $w \in \Sigma^*$  into a bipartite graph  $G = (V_1, V_2, E)$  with  $V_1 = [|w|]$  and  $V_2 = \Sigma$ . Lemmas 14 then produce an equivalent PCIO-instance with an associated partial order  $\rho_w$  on  $V_2$  that is an interval order. For two letters  $a, b \in \Sigma$ ,  $(a, b) \in \rho_w$  means that the last occurrence of  $a$  in  $w$  comes before the first occurrence of  $b$  in  $w$ . Obviously,  $SCD(w)$  is the maximum size of an anti-chain in  $\rho_w$ . Hence, the previously mentioned results of Habib and Möhring imply, together with Lemma 5:

► **Lemma 30.**  $SCD(w) = pw(G_{\rho_w}) + 1 = cpw(G_{\rho_w}, \rho_w) + 1$ .

Theorem 1 has therefore the following consequences for the string parameter  $SCD$ . To the best of our knowledge, this is the first algorithmic exploit of this string parameter.

► **Corollary 31.** *GBS can be solved in  $\mathcal{O}^*(SCD(w)2^{SCD(w)})$ .*

As the scope coincidence degree of a word  $w \in \Sigma^*$  is upper-bounded by  $|\Sigma|$ , we also obtain the following result for the parameter  $|\Sigma|$  that improves on Lemma 24.

► **Corollary 32.** *GBS can be solved in  $\mathcal{O}^*(|\Sigma|2^{|\Sigma|})$ .*

There is another graph-theoretic interpretation of the scope coincidence degree presented by Reidenbach and Schmid [49] for patterns. It relates to our setting as follows. To a string  $w \in \Sigma^n$ , we associate its *Gaifman graph*  $\Gamma_w$  with vertex set  $[n]$  and edges  $(i, i + 1)$  for  $i \in [n - 1]$ , as well as the edge sets  $E_a = \{(\min Scope(a), j) \mid j \in Scope(a)\}$  (disregarding loops) for each  $a \in \Sigma$ . According to [49, Lemma 15],  $pw(\Gamma_w) \leq SCD(w) + 1$ . It might be interesting to further link the pathwidths of  $\Gamma_w$  and of  $G_{\rho_w}$ . Do they differ by exactly two?

Inspired by the considerations on the *range of a candidate* in KRA, the *maximum scope*  $s_{\max} \doteq \max_{a \in \Sigma} |\text{Scope}(a)|$  could be another parameterization for GBS. Similar to Lemma 21, one can show that GBS, parameterized by  $s_{\max}$ , is in FPT. It would also be meaningful to interpret this parameter in the context of OSCM for graph visualization reasons.

## 7 Conclusion

Finally, we explain some further connections and future lines of research. Recall that we did list several concrete open problems throughout the paper that we are not going to repeat here, but they are clearly also natural continuations of the present study.

**Different types of partial orderings.** It would be interesting to have a closer look to different types of partial orderings in the context of PCO. For instance, the papers of Brandenburg and Gleißer [8] or Hudry [34] list quite a lot of different types of partial orders (in the context of rank aggregation problems). We can also view this research as a starting point to systematically look at decision problems related to partial orders from the viewpoint of parameterized complexity. Then, [7] might be a good starting point.

**Related problems, popular with Operations Research.** In the Operations Research Community, there has also been lots of studies of the *linear ordering polytope*. Regarding the problems studied in this paper, [10] might be a good starting point. Likewise, the so-called OPTIMAL LINEAR EXTENSION PROBLEM has been considered in the literature [41]. However, only the costs of the immediate neighborhood in the target linear order are considered, similar to the famous TRAVELLING SALESPERSON PROBLEM,<sup>5</sup> while we sum up all costs associated to pairs  $(x, y)$  with  $x < y$  in the final linear order  $<$ .

**Putting additional constraints: a theme arising in Graph Drawing and in Order Theory.** Forster [27] argues that the CONSTRAINED OSCM problem, where a partial order on  $V_2$  is given in addition, that should be extended to a linear ordering (as before), has quite some applications. This can be clearly modeled as an instance of CO, but some further research is needed to conclude the same type of results as we did for OSCM with the interval order approach. This might relate to earlier (systematic) research on the realizability of constraints on interval orders, see [47, 48]. In particular the distance constraints might be indeed interesting for graph drawing purposes, as the neighbor vertices should not stretch out too much.

**Remarks on approximation.** For the minimization problem related to PCO, a PTAS is known according to [25]. Our reasoning immediately implies the existence of PTAS for OSCM, KRA and GBS. In view of the tedious factor-1.4664 approximation for OSCM presented in [46], this shows again the strength of looking at these specific problems from a wider perspective.

**Comments on approximation and heuristics.** We suggest that the tight connections that we found between GBS and OSCM should also be interesting in the development and analysis of (heuristic) algorithms for both problems. In this context, it is interesting to observe that

---

<sup>5</sup> The difference between cycles (tours) and paths do not matter for the involved algorithms that much.

Wong and Reingold [55] proposed a median heuristic for computing a solution to a given GBS instance. They proved that on random instances, this heuristic is at most 10% off from the optimum (in expectation). Moreover, the larger random instances are picked, the smaller is the relative error of the median heuristic (in expectation). Incidentally, the same (median) heuristic was suggested by Eades and Wormalds [21] some years later for OSCM. They proved that this heuristic is a factor-3 approximation, but did not go into a randomized analysis. Our translation of GBS into OSCM actually proves the following which is the last result of this paper.

► **Corollary 33.** *The median heuristic gives a factor-3 approximation for GBS.*

---

## References

- 1 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009. doi:10.1007/978-3-642-02927-1\_6.
- 2 J. Bartholdi, III, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.
- 3 Oliver Bastert and Christian Matuszewski. Layered drawings of digraphs. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*, pages 87–120. Springer, 1999. doi:10.1007/3-540-44969-8\_5.
- 4 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
- 5 Nadja Betzler, Michael R. Fellows, Jiong Guo, Rolf Niedermeier, and Frances A. Rosamond. Fixed-parameter algorithms for kemeny rankings. *Theor. Comput. Sci.*, 410(45):4554–4570, 2009. doi:10.1016/j.tcs.2009.08.033.
- 6 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
- 7 Vincent Bouchitté and Michel Habib. NP-completeness properties about linear extensions. *Order*, 4(2):143–154, 1987.
- 8 Franz J. Brandenburg and Andreas Gleißner. Ranking chain sum orders. *Theor. Comput. Sci.*, 636:66–76, 2016. doi:10.1016/j.tcs.2016.05.026.
- 9 Franz-Josef Brandenburg, Andreas Gleißner, and Andreas Hofmeier. Comparing and aggregating partial orders with kendall tau distances. *Discrete Math., Alg. and Appl.*, 5(2), 2013. doi:10.1142/S1793830913600033.
- 10 Christoph Buchheim, Angelika Wiegele, and Lanbo Zheng. Exact algorithms for the quadratic linear ordering problem. *INFORMS J. Comput.*, 22(1):168–177, 2010. doi:10.1287/ijoc.1090.0318.
- 11 Olca A. Çakiroglu, Cesim Erten, Ömer Karatas, and Melih Sözdinler. Crossing minimization in weighted bipartite graphs. *J. Discrete Algorithms*, 7(4):439–452, 2009. doi:10.1016/j.jda.2008.08.003.
- 12 Irène Charon and Olivier Hudry. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR*, 5(1):5–60, 2007. doi:10.1007/s10288-007-0036-6.
- 13 Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM J. Comput.*, 42(3):792–807, 2013. doi:10.1137/11083856X.
- 14 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discret. Math.*, 23(4):1905–1953, 2009. doi:10.1137/S0895480100373455.

- 15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 16 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In Giuseppe Liotta, editor, *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*, pages 332–344. Springer, 2003. doi:10.1007/b94919.
- 17 Vida Dujmovic, Henning Fernau, and Michael Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323, 2008.
- 18 Vida Dujmovic and Sue Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004.
- 19 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 613–622. ACM, 2001. doi:10.1145/371920.372165.
- 20 Peter Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21A:89–98, 1986.
- 21 Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.
- 22 Eduard Eiben, Robert Ganian, Kustaa Kangas, and Sebastian Ordyniak. Counting linear extensions: Parameterizations by treewidth. *Algorithmica*, 81(4):1657–1683, 2019. doi:10.1007/s00453-018-0496-4.
- 23 Henning Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Universität Tübingen, Germany, 2005.
- 24 Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Matthias Mnich, Geevarghese Philip, and Saket Saurabh. Ranking and drawing in subexponential time. In Costas S. Iliopoulos and William F. Smyth, editors, *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers*, volume 6460 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2011.
- 25 Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Matthias Mnich, Geevarghese Philip, and Saket Saurabh. Social choice meets graph drawing: How to get subexponential time algorithms for ranking and drawing problems. *TSINGHUA SCIENCE AND TECHNOLOGY*, 19(4):374–386, 2014.
- 26 Peter C. Fishburn. *Interval Orders and Interval Graphs*. John Wiley, 1985.
- 27 Michael Forster. A fast and simple heuristic for constrained two-level crossing reduction. In János Pach, editor, *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers*, volume 3383 of *Lecture Notes in Computer Science*, pages 206–216. Springer, 2004. doi:10.1007/978-3-540-31843-9\_22.
- 28 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 29 Paul C Gilmore and Alan J Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.
- 30 Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.
- 31 Michel Habib and Rolf H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11(1):47–60, 1994.
- 32 Patrick Healy and Nikola S. Nikolov. Hierarchical drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 409–453. Chapman and Hall/CRC, 2013.

- 33 Wen-Lian Hsu and Tze-Heng Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Comput.*, 28(3):1004–1020, 1999. doi:10.1137/S0097539792224814.
- 34 Olivier Hudry. NP-hardness results for the aggregation of linear orders into median orders. *Annals of Operations Research*, 163:63–88, 2008.
- 35 Marek Karpinski and Warren Schudy. Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2010.
- 36 John G. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.
- 37 Yasuaki Kobayashi, Hirokazu Maruta, Yusuke Nakae, and Hisao Tamaki. A linear edge kernel for two-layer crossing minimization. *Theor. Comput. Sci.*, 554:74–81, 2014.
- 38 Yasuaki Kobayashi and Hisao Tamaki. A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790, 2015.
- 39 Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representations in logspace. *SIAM J. Comput.*, 40(5):1292–1315, 2011. doi:10.1137/10080395X.
- 40 Johannes Köbler, Sebastian Kuhnert, and Osamu Watanabe. Interval graph representation with given interval and intersection lengths. *J. Discrete Algorithms*, 34:108–117, 2015. doi:10.1016/j.jda.2015.05.011.
- 41 Longcheng Liu, Biao Wu, and Enyu Yao. Minimizing the sum cost in linear extensions of a poset. *J. Comb. Optim.*, 21(2):247–253, 2011. doi:10.1007/s10878-009-9237-6.
- 42 Roy Lowrance and Robert A. Wagner. An extension of the string-to-string correction problem. *J. ACM*, 22(2):177–183, 1975. doi:10.1145/321879.321880.
- 43 B. Monjardet. Tournois et ordres médians pour une opinion. *Mathématiques et Sciences humaines*, 43:55–73, 1973.
- 44 Xavier Muñoz, Walter Unger, and Imrich Vrt’o. One sided crossing minimization is NP-hard for sparse graphs. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, volume 2265 of *Lecture Notes in Computer Science*, pages 115–123. Springer, 2001. doi:10.1007/3-540-45848-4.
- 45 Petra Mutzel. Optimization in leveled graphs. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization, Second Edition*, pages 2813–2820. Springer, 2009. doi:10.1007/978-0-387-74759-0\_483.
- 46 Hiroshi Nagamochi. An improved bound on the one-sided minimum crossing number in two-layered drawings. *Discret. Comput. Geom.*, 33(4):569–591, 2005. doi:10.1007/s00454-005-1168-0.
- 47 Itsik Pe’er and Ron Shamir. Realizing interval graphs with size and distance constraints. *SIAM J. Discret. Math.*, 10(4):662–687, 1997. doi:10.1137/S0895480196306373.
- 48 Itsik Pe’er and Ron Shamir. Satisfiability problems on intervals and unit intervals. *Theor. Comput. Sci.*, 175(2):349–372, 1997. doi:10.1016/S0304-3975(96)00208-3.
- 49 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Inf. Comput.*, 239:87–99, 2014.
- 50 Carl Sechen. *VLSI placement and global routing using simulated annealing*, volume 54. Springer Science & Business Media, 2012.
- 51 Narges Simjour. Improved parameterized algorithms for the Kemeny aggregation problem. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2009.

- 52 Matthias F. M. Stallmann, Franc Brglez, and Debabrata Ghosh. Heuristics, experimental subjects, and treatment evaluation in bigraph crossing minimization. *ACM J. Exp. Algorithmics*, 6:8, 2001. doi:10.1145/945394.945402.
- 53 Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.
- 54 William T. Trotter. New perspectives on interval orders and interval graphs. In R. A. Bailey, editor, *Surveys in Combinatorics*, volume 241 of *London Mathematical Society Lecture Note Series*, pages 237–286. 1997.
- 55 D. F. Wong and Edward M. Reingold. Probabilistic analysis of a grouping algorithm. *Algorithmica*, 6(2):192–206, 1991. doi:10.1007/BF01759041.
- 56 H. P. Young and A. Levenglick. A consistent extension of Condorcet’s election principle. *SIAM J. Appl. Math.*, 35(2):285–300, 1978.
- 57 Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. doi:10.1287/moor.1090.0385.

# Optimal Output Sensitive Fault Tolerant Cuts

**Niranka Banerjee**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
nirankab@imsc.res.in

**Venkatesh Raman**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
vraman@imsc.res.in

**Saket Saurabh**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
University of Bergen, Norway  
saket@imsc.res.in

---

## Abstract

In this paper we consider two classic cut-problems, GLOBAL MIN-CUT and MIN  $k$ -CUT, via the lens of fault tolerant network design. In particular, given a graph  $G$  on  $n$  vertices, and a positive integer  $f$ , our objective is to compute an upper bound on the size of the sparsest subgraph  $H$  of  $G$  that preserves edge connectivity of  $G$  (denoted by  $\lambda(G)$ ) in the case of GLOBAL MIN-CUT, and  $\lambda(G, k)$  (denotes the minimum number of edges whose removal would partition the graph into at least  $k$  connected components) in the case of MIN  $k$ -CUT, upon failure of any  $f$  edges of  $G$ . The subgraph  $H$  corresponding to GLOBAL MIN-CUT and MIN  $k$ -CUT is called  $f$ -FTCS and  $f$ -FT- $k$ -CS, respectively. We obtain the following results about the sizes of  $f$ -FTCS and  $f$ -FT- $k$ -CS.

- There exists an  $f$ -FTCS with  $(n - 1)(f + \lambda(G))$  edges. We complement this upper bound with a matching lower bound, by constructing an infinite family of graphs where any  $f$ -FTCS must have at least  $\frac{(n - \lambda(G) - 1)(\lambda(G) + f - 1)}{2} + (n - \lambda(G) - 1) + \frac{\lambda(G)(\lambda(G) + 1)}{2}$  edges.
- There exists an  $f$ -FT- $k$ -CS with  $\min\{(2f + \lambda(G, k) - (k - 1))(n - 1), (f + \lambda(G, k))(n - k) + \ell\}$  edges. We complement this upper bound with a lower bound, by constructing an infinite family of graphs where any  $f$ -FT- $k$ -CS must have at least  $\frac{(n - \lambda(G, k) - 1)(\lambda(G, k) + f - k + 1)}{2} + n - \lambda(G, k) + k - 3 + \frac{(\lambda(G, k) - k + 3)(\lambda(G, k) - k + 2)}{2}$  edges.

Our upper bounds exploit the structural properties of  $k$ -connectivity certificates. On the other hand, for our lower bounds we construct an infinite family of graphs, such that for any graph in the family any  $f$ -FTCS (or  $f$ -FT- $k$ -CS) must contain all its edges. We also add that our upper bounds are constructive. That is, there exist polynomial time algorithms that construct  $H$  with the aforementioned number of edges.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis; Theory of computation → Sparsification and spanners

**Keywords and phrases** Fault tolerant, minimum cuts, upper bound, lower bound

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.10

**Funding** *Saket Saurabh*: Received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.



## 1 Introduction

There is a common proverb in English – *m it is better to be safe than sorry!* Probably, it has never been more true than the Covid-19-times we are living in. Closed in our homes, computers are probably our only way of communicating with the world. Our machines are part of a larger network – it is just a node in the network. Thus, to get past this moment in time we need our networks to be more reliable, than ever before. Unfortunately, most of



© Niranka Banerjee, Venkatesh Raman, and Saket Saurabh;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 10; pp. 10:1–10:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the real life networks are prone to failures. A failure of a link (or a small number of links) in the network may lead to a breakdown in communication. This motivates us to build networks that are resilient to failures, leading to the field of *fault tolerant network design*.

Networks are best modelled as graphs. For example, we could imagine we have a communication network, where the nodes (or vertices) are computers, routers, or cell-towers and there is an edge between them if they can communicate. One could also imagine a transportation network, where the edges correspond to segment of a road and the junctions between the roads are vertices. Once we have abstracted these networks as graphs, there are a number of properties we could try to ask about graphs that are meaningful for the particular network they represent. As stated earlier, real life networks are prone to failures. That is, edges (or vertices) may change their status from active to failed, and vice versa. These failures may occur anytime; however it is expected that they are small in numbers. Further, we can assume that failures are not permanent as they are repaired simultaneously. The fact that we only have a small number of failures is captured by associating an integer—a fault parameter  $f$  with the network. That is, we assume that at any point of time we only have at most  $f$ -edges (or vertices) that are failed. Indeed,  $f$  is much smaller than the number of vertices in the graph. This motivates the research on designing fault tolerant structures for various graph problems in terms of fault parameter  $f$  and the input size  $n$ .

We now formally define the model of *fault tolerant network design*, with respect to a property  $\Pi$ , we would be interested in. A property of graphs is a function  $\sigma$  that assigns to each graph a value in  $\{\text{true}, \text{false}\}$ . Given a graph  $G$ , a fault parameter  $f$ , we want to find a subgraph  $H$  of  $G$ , such that for any set  $F \subseteq E(G) \setminus E(H)$  of size  $f$ , we have the following:  $\sigma(G - F)$  is true if and only if  $\sigma(H - F)$  is true. In general, the solution of a fault tolerant network design is measured by the size of the subgraph  $H$ . That is, our objective is to find  $H$  with as few edges as possible. Fault tolerant subgraphs have been developed for various problems like reachability [3, 4, 8], shortest path [6, 20, 37–39] and spanners [5, 7, 9, 12, 36]. A fault tolerant subgraph for single source reachability in directed graphs was shown by Baswana et al. [4] to contain  $\Theta(2^f n)$  edges. Given a graph  $G$ , a source  $s$ , and an integer  $f$ , a subgraph  $H$  is an  $(\alpha, \beta)$ -single source fault tolerant subgraph, if for every vertex  $v \in V(G)$ , for every  $F \subseteq E(G)$  of size at most  $f$ ,  $\text{dist}(s, v, H - F) \leq \alpha \cdot \text{dist}(s, v, G - F) + \beta$ . Parter and Peleg [39] gave an  $(3(f + 1), (f + 1) \log n)$ -single source fault tolerant subgraph with  $\mathcal{O}(fn)$  edges. For spanners with a stretch  $k$ , Dinitz et al. [12] gave an  $f$ -fault tolerant  $k$ -spanner with  $\tilde{\mathcal{O}}(f^2 n^{1 + \frac{2}{k+1}})$  edges. Recently, Chakraborty and Choudhary [8] showed an  $\mathcal{O}(n + \min\{|P|, \sqrt{n}\}, n\sqrt{|P|})$  bound on a subgraph, that is an 1-fault tolerant reachability preserver for a given vertex-pair set  $P \subseteq V(G) \times V(G)$ .

Our main objective of this article is to extend this study to two classic cut-problems, GLOBAL MIN-CUT and MIN  $k$ -CUT. Arguably, GLOBAL MIN-CUT and MIN  $k$ -CUT are one of the two most well-studied problems in the field of graph algorithms. In the MIN  $k$ -CUT problem, input is an undirected graph  $G$  and an integer  $k$ , and the task is to partition the vertex set into  $k$  non-empty sets, say  $\tilde{P}$ , such that the total number of the edges with endpoints in different parts is minimized. We call such a partition as *min  $k$ -cut*, or simply a  *$k$ -cut*. For  $k = 2$ , rather than saying *2-cut*, we say *min-cut*. Indeed, for  $k = 2$ , this is the classic GLOBAL MIN-CUT problem, which can be solved in polynomial time. In fact, for every fixed  $k$ , the problem is known to be polynomial time solvable [18]. However, when  $k$  is part of the input, the problem is NP-complete [18]. Both these problems have been extensively studied in the last 30 years, and the running time of algorithms for these two problems have been improved over the years [10, 15, 19, 22, 24–28, 30, 32, 35, 40, 41]. In particular, after a series of improvement, the fastest known algorithm for GLOBAL MIN-CUT in unweighted

graphs is given by Ghaffari et al. [17] that runs in time  $\mathcal{O}(m \log n)$ . On the other hand, for edge-weighted graphs the fastest known algorithm for GLOBAL MIN-CUT is independently given by Gawrychowski et al [16] and Mukhopadhyay and Nanongkai [34] and (almost) runs in time  $\mathcal{O}(m \log^2 n)$ . Both of these algorithms are randomized. The best known deterministic algorithm for the problem on unweighted graph is given by Henzinger et al. [23] and runs in time  $\mathcal{O}(m \log^2 n (\log \log n)^2)$ .

The history of MIN  $k$ -CUT problem is also extremely rich. The direction of polynomial time approximation algorithms is essentially settled, with factor  $2(1 - \frac{1}{k})$  approximation algorithms and matching lower bounds. Recently, Gupta et al. [19] showed that for every fixed  $k \geq 2$ , the Karger-Stein algorithm [29] outputs any fixed  $k$ -cut with probability at least  $\widehat{\mathcal{O}}(n^{-k})$ , where  $\widehat{\mathcal{O}}(\cdot)$  hides a  $2^{\mathcal{O}(\ln \ln n)^2}$  factor. This immediately gives an extremal bound of  $\widehat{\mathcal{O}}(n^k)$ , on the number of minimum  $k$ -cuts in an  $n$ -vertex graph and an algorithm for MIN  $k$ -CUT in similar running time. Both the extremal bound and the running time of the algorithm are essentially tight (under reasonable assumptions). Indeed the extremal bound matches known lower bounds up to  $\widehat{\mathcal{O}}(1)$  factors, while any further improvement to the exact algorithm would imply an improved algorithm for MAX-WEIGHT  $k$ -CLIQUE [1, 2], which has been conjectured not to exist. One can also obtain  $f(k)n^{o(k)}$  lower bound on the running time [11, 13] under the Exponential Time Hypothesis (ETH). In the world of FPT-approximation, MIN  $k$ -CUT is known to admit  $(1 + \epsilon)$  approximation algorithm running in time  $(\frac{k}{\epsilon})^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$  [31].

## 1.1 Our Results and Methods

In this paper we initiate a new research direction to the studies of GLOBAL MIN-CUT and MIN  $k$ -CUT. In particular we do the following.

We focus on GLOBAL MIN-CUT and MIN  $k$ -CUT, via the lens of fault tolerant network design, and construct asymptotically optimal fault tolerant subgraphs for these two problems.

Given a graph  $G$ , let  $\lambda(G)$  and  $\lambda(G, k)$  denote the size of min-cut and  $k$ -cut of  $G$ , respectively. We formally define the objects we consider in the paper.

► **Definition 1.1** (*f*-FTCS (*f*-FT- $k$ -CS)). *An f-FTCS (f-FT- $k$ -CS) is a subgraph  $H$  of  $G$  such that for any set of edges  $F \subseteq E(G)$  of cardinality at most  $f$ ,  $\lambda(G - F) = \lambda(H - F)$  ( $\lambda(G - F, k) = \lambda(H - F, k)$ ). For a graph  $G$ , we use  $\Psi(G, k)$  to denote the minimum number of edges in a *f*-FT- $k$ -CS of  $G$ . That is,*

$$\Psi(G, k) = \min_{H \text{ is an } f\text{-FT-}k\text{-CS of } G} |E(H)|$$

When  $k = 2$ , this denotes the minimum number of edges in a *f*-FTCS of  $G$ . In this case we simply use  $\Psi(G)$ , rather than  $\Psi(G, 2)$ .

Let  $\mathcal{F}$  be a family of graphs, then for all  $n \in \mathbb{N}$ , we define the following:

$$\begin{aligned} \text{FTCS}(\mathcal{F}, n, f) &= \max_{G \in \mathcal{F}, |V(G)|=n} \Psi(G) \\ \text{FT-}k\text{-CS}(\mathcal{F}, n, f) &= \max_{G \in \mathcal{F}, |V(G)|=n} \Psi(G, k) \end{aligned}$$

When  $\mathcal{F}$  is the family of all graphs, then we simply use  $\text{FTCS}(n, f)$  and  $\text{FT-}k\text{-CS}(n, f)$ . Our goal is to give asymptotic upper bounds on  $\text{FTCS}(n, f)$  and  $\text{FT-}k\text{-CS}(n, f)$ . Since any

## 10:4 Optimal Output Sensitive Fault Tolerant Cuts

graph has at most  $\binom{n}{2}$  edges, we have that  $\text{FTCS}(n, f)$  (or  $\text{FT-}k\text{-CS}(n, f)$ ) is at most  $\mathcal{O}(n^2)$ . Let  $G$  be a clique on  $n$  vertices. First, note that  $\lambda(G) = n - 1$ . Next observe that any  $f$ -FTCS,  $H$  of  $G$ , even for  $f = 1$ , must contain *all the edges of the clique*. Indeed, if an edge  $(u, v) \in E(G)$  is not present in  $H$ , then the adversary may delete an edge adjacent to  $u$  or  $v$  in the clique, that is not  $(u, v)$ . In this case,  $\lambda(G - F) = n - 2$ , whereas  $\lambda(H - F) \leq n - 3$ . This simple construction shows that  $\text{FTCS}(n, 1)$  is at least  $\Omega(n^2)$ . This bound tells us that for these problems we can not improve upon the trivial upper bounds.

Our example with family of cliques seems to suggest that we have reached the end of the road. However, on the second look we observe that for a clique even  $\lambda(G) = \Omega(n)$ . Thus, we can also express our lower bound as  $\lambda(G) \cdot n$ . This motivates us to look for a *fine-grained* definition of  $\text{FTCS}(n, f)$  and  $\text{FT-}k\text{-CS}(n, f)$ , that not only takes into account  $n$  and  $f$ , but also some parameter that captures the edge-connectivity (or the value of  $k$ -cut) of the input graph. In particular, we can come up with the following new definitions. Let  $\mathcal{F}$  be a family of graphs, then for all  $n, \ell \in \mathbb{N}$ , we define the following:

$$\begin{aligned} \text{FTCS}(\mathcal{F}, n, \ell, f) &= \max_{G \in \mathcal{F}, |V(G)|=n, \lambda(G)=\ell} \Psi(G) \\ \text{FT-}k\text{-CS}(\mathcal{F}, n, \ell, f) &= \max_{G \in \mathcal{F}, |V(G)|=n, \lambda(G, k)=\ell} \Psi(G, k) \end{aligned}$$

With respect to our new definition, when  $\mathcal{F}$  is a family of cliques, we have that  $\text{FTCS}(\mathcal{F}, n, \ell, 1)$  is at most  $\mathcal{O}(\ell n)$ . Thus, a natural question arises: Can we derive similar upper bound even when  $\mathcal{F}$  denotes the family of all graphs? Indeed, we provide a matching upper and lower bound on these quantities in this paper. As before, when  $\mathcal{F}$  is the family of all graphs. Then, we simply use  $\text{FTCS}(n, \ell, f)$  and  $\text{FT-}k\text{-CS}(n, \ell, f)$ . Our first result is the following.

► **Theorem 1.2.** *Let  $n, \ell$  and  $f$  be three positive integers. Then,  $\text{FTCS}(n, \ell, f)$  is upper bounded by  $(f + \ell)(n - 1)$ .*

The proof of Theorem 1.2 is inspired from the concept of *k-connectivity certificates* used in the literature [14, 35]. For a  $k$ -edge connected graph  $G = (V, E)$ , a subset of edges  $E' \subseteq E$  is called a  $k$ -connectivity certificate of the graph  $G$ , if the subgraph  $G' = (V, E')$  is  $k$ -edge connected. For a  $k$ -edge connected graph on  $n$  vertices, there always exists a  $k$ -connectivity certificate with at most  $k(n - 1)$  edges [14]. For our proof, we modify a known construction of a  $k$ -connectivity certificate to also handle edge failures.

Our second result complements the above upper bound, by showing that this bound is tight upto constant factors. Specifically, we show the following.

► **Theorem 1.3.** *There exists an infinite family of triplets  $(n, \ell, f)$  such that*

$$\text{FTCS}(n, \ell, f) \geq \frac{(n - \ell - 1)(\ell + f - 1)}{2} + (n - \ell - 1) + \frac{\ell(\ell + 1)}{2}.$$

To prove Theorem 1.3, we construct an infinite family of graphs  $(\mathcal{G})$ , such that for any  $G \in \mathcal{G}$  we have that any  $f$ -FTCS of  $G$  must contain all its edges. In particular, for any positive integers  $n, \ell, f$ , such that  $\frac{n - \ell - 1}{\ell + f}$  is an *integer*, we construct a graph  $G$  on  $n$  vertices and  $(n - \ell - 1)\frac{(\ell + f - 1)}{2} + (n - \ell - 1) + \frac{\ell(\ell + 1)}{2}$  edges with  $\lambda(G) = \ell$  (note that  $\ell \leq n - 1$ ) such that any  $f$ -FTCS of  $G$  must contain all the edges of  $G$ . The construction of the family  $\mathcal{G}$ , and the analysis that for any graph  $G \in \mathcal{G}$ , any  $f$ -FTCS of  $G$  must contain all its edges are quite technical.

Next we generalize our results on GLOBAL MIN-CUT to MIN  $k$ -CUT and give the following two results about  $\text{Ft-}k\text{-cs}(n, \ell, f)$ .

► **Theorem 1.4.** *Let  $n, \ell$  and  $f$  be three positive integers. Then,  $\text{FT-}k\text{-CS}(n, \ell, f)$  is upper bounded by  $\min\{2f + \ell - (k - 1)(n - 1), (f + \ell)(n - k) + \ell\}$ .*

Proof of Theorem 1.4 is quite involved and requires understanding the intricate relationship between edge-connectivity certificates and the MIN  $k$ -CUT problem. This is one of the main technical results. In our final result, we complement Theorem 1.4 with a tight lower bound.

► **Theorem 1.5.** *There exists an infinite family of triplets  $(n, \ell, f)$  such that*

$$\text{FT-}k\text{-CS}(n, \ell, f) \geq \frac{(n - \ell - 1)(\ell + f - k + 1)}{2} + n - \ell + k - 3 + \frac{(\ell - k + 3)(\ell - k + 2)}{2}.$$

While the construction is somewhat similar in spirit to the construction of the lower bound for the construction of the family of graphs for GLOBAL MIN-CUT, the proof of correctness is even more involved.

**Tightness of our Upper and Lower Bounds.** Notwithstanding the fact that the leading terms in our upper and lower bounds appear close, there are some negative quantities in the leading terms, and in some ranges, the other terms in the bounds dominate. Still, our bounds for GLOBAL MIN-CUT are asymptotically optimal. For example in the lower bound for GLOBAL MIN-CUT (Theorem 1.3), when  $n - \ell$  becomes  $o(n)$ ,  $\ell$  is  $\Omega(n)$  and in this case the  $\frac{\ell(\ell+1)}{2}$  bound dominates and we get a lower bound of  $\Omega(n^2)$  which is asymptotically optimal given our upper bound and the range of  $\ell$ . When  $\ell$  is  $o(n)$ , our lower bound is  $\Omega((f + \ell)n)$  which matches asymptotically with the upper bound.

For MIN  $k$ -CUT however, there are some gaps. For example, if  $\ell = n - 1$  and  $k = n - \log n$ , the upper bound is  $\mathcal{O}((f + n) \log n)$  but the lower bound is  $\Omega(n)$ . Such a gap exists in some ranges of  $f$  and  $k$  when  $n - \ell$  and  $\ell - k$  are both  $o(n)$ . However, when  $n - \ell$  or  $\ell - k$  is  $\Theta(n)$ , our upper and lower bounds are a constant factor away from each other.

**Algorithmic Considerations.** The proof of Theorem 1.2 is constructive. That is, given a graph  $G$  and an integer  $f$ , in polynomial time we can construct an  $f$ -FTCS of  $G$  with at most  $(f + \lambda(G))(n - 1)$  edges. For this algorithm we just need the value of  $\lambda(G)$ , which can be computed in  $\mathcal{O}(m \log^2 n (\log \log n)^2)$  time [23]. However, the proof of Theorem 1.4 is “almost” constructive. That is, the proof can be made constructive, if for a graph  $G$  we can compute the value of  $\lambda(G, k)$  in polynomial time. Indeed, for a constant value of  $k$ , we could use the polynomial time algorithm running in time  $n^{\mathcal{O}(k)}$  [10, 19, 41]. However, the running time of this algorithm grows with  $k$ , and hence becomes prohibitive quite soon. Thus, as an alternative we could use an upper bound on  $\lambda(G, k)$ , provided by the known polynomial time factor 2 approximation algorithm [21, 42]. This leads to an upper bound of  $\min\{2f + 2\lambda(G, k) - (k - 1)(n - 1), (f + 2\lambda(G, k))(n - k) + 2\lambda(G, k)\}$  on the constructed  $f$ -FT- $k$ -CS, which is slightly worse than the upper bound provided by Theorem 1.4.

## 2 Preliminaries

Given an integer  $q$ , we use  $[q]$  to denote  $\{1, \dots, q\}$ . Further, for two integers,  $q_1 \leq q_2$ , we use  $[q_1, q_2]$  to denote  $\{q_1, \dots, q_2\}$ . For a graph  $G = (V, E)$ , we also use  $V(G)$  and  $E(G)$  to denote the set of vertices and the set of edges of graph  $G$ , respectively. A path  $P$  in  $G$  is a sequence of distinct vertices ( $P = v_1 v_2 \dots v_q$ ), such that two consecutive vertices have an edge between them. Let  $A_1, \dots, A_\ell$  be a partition of the vertex set  $V(G)$  of a graph  $G$ . That is,  $\cup_{i=1}^{\ell} A_i = V(G)$  and for all  $i \neq j$ ,  $A_i \cap A_j = \emptyset$ . We use  $E(A_1, \dots, A_\ell, G)$  to denote the set

## 10:6 Optimal Output Sensitive Fault Tolerant Cuts

of edges such that each edge in the set has one endpoint in  $A_i$  and the other endpoint in  $A_j$ , where  $i \neq j$ . For a graph  $G$ , and a pair of vertices  $u, v \in V(G)$ , we use  $\lambda_G(u, v)$  to denote the minimum number of edges whose removal separates  $u$  and  $v$  (that is,  $u$  and  $v$  belong to different connected components). If the graph  $G$  is clear from the context, we omit the subscript  $G$  from  $\lambda_G(u, v)$ , and simply write  $\lambda(u, v)$ . Next, we state the classical Menger's Theorem and a simple lemma which are crucially used in our proofs,

► **Lemma 2.1** (Menger's Theorem, [33]). *Let  $G$  be an undirected graph and let  $u$  and  $v$  be two vertices of  $G$ . Then the maximum number of pairwise edge-disjoint  $u$ - $v$  paths in  $G$  is equal to  $\lambda(u, v)$ .*

► **Lemma 2.2.** *Let  $G = (V, E)$  be an undirected graph and let  $H$  be a subgraph of  $G$ . Let  $k > 1$  be an integer. Then,  $\lambda(H) \leq \lambda(G)$  and  $\lambda(H, k) \leq \lambda(G, k)$ .*

### 3 Global Min-Cut

In this section we develop upper and lower bounds on  $\text{FTCS}(n, \ell, f)$ . In particular we prove Theorems 1.2 and 1.3.

#### 3.1 Upper Bound

Let  $n, \ell$  and  $f$  be three positive integers. We need to show that  $\text{FTCS}(n, \ell, f)$  is upper bounded by  $(f + \ell)(n - 1)$ . Towards this we show that given an undirected graph  $G$ , and an integer  $f$ , we can construct an  $f$ -FTCS,  $H$ , of  $G$  on at most  $(f + \lambda(G))(n - 1)$  edges. Indeed, when  $\lambda(G) = \ell$ , the upper bound follows. Further, we assume  $G$  is connected. If  $G$  is disconnected then  $\lambda(G) = 0$ , and it remains so after any edge failure. Thus, in this case we can take  $H$  to be an empty graph. Our construction is presented next.

##### Construction of an $f$ -FTCS of a graph $G$ .

1. Initialize  $f + \ell$  empty (no edges) forests  $T_1, T_2, \dots, T_{f+\ell}$  on the same vertex set  $V(G)$ .
2. for each edge  $(u, v) \in E(G)$ , do the following.
  - Find the smallest integer  $i \in [f + \ell]$ , such that  $u$  and  $v$  are in different connected components of  $T_i$ . If no such  $i$  exists, then assign  $i$  to  $\infty$ .
  - If  $i$  is not  $\infty$  then add  $(u, v)$  to  $T_i$ .
3. Output  $H = \cup_{a=1}^{f+\ell} T_a$ .

We will show that  $H$  is an  $f$ -FTCS with at most  $(f + \ell)(n - 1)$  edges. The bound on the number of edges on  $H$  is clear, as  $H$  is the union of at most  $(f + \ell)$  forests.

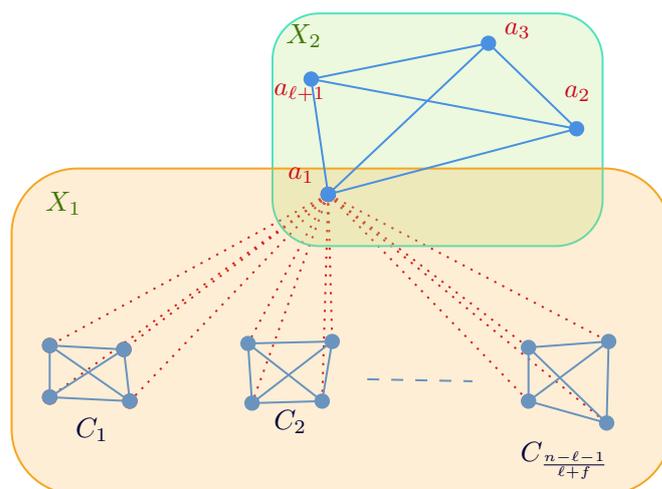
► **Lemma 3.1.** *The subgraph  $H$  has at most  $(f + \ell)(n - 1)$  edges.*

Next, we show that  $H$  is an  $f$ -FTCS. We start with the following observation.

► **Lemma 3.2.**  $(\star)^1$  *Let  $(u, v) \in E(G) \setminus E(H)$ . Then there are at least  $\ell + f$  edge-disjoint paths between  $u$  and  $v$  in  $G$  and  $H$ .*

---

<sup>1</sup> Results marked with  $\star$  are deferred to the full version.



■ **Figure 1** Vertex  $a_1$  has degree  $n - \ell - 1$  within  $X_1$ . Vertices  $a_1, a_2, \dots, a_{\ell+1}$  in  $X_2$  form an induced  $\ell + 1$ -clique.  $C_1, C_2, \dots, C_{\frac{n-\ell-1}{\ell+f}}$  represent  $\frac{n-\ell-1}{\ell+f}$  cliques each of size  $\ell + f$  within  $X_1$ .

To prove that  $H$  is an  $f$ -FTCS of  $G$ , we need to show that for any set of edges  $F \subseteq E(G)$  of cardinality at most  $f$ ,  $\lambda(H - F) = \lambda(G - F)$ . As  $H$  is a subgraph of  $G$ , we know from Lemma 2.2 that  $\lambda(H - F) \leq \lambda(G - F)$ . Now we show that  $\lambda(H - F) \geq \lambda(G - F)$ .

► **Lemma 3.3.** *Let  $G$  be an undirected graph with  $\lambda(G) = \ell$ ,  $f$  be a positive integer, and  $H$  be the subgraph constructed above. Then for any set  $F$  of at most  $f$  edges,  $\lambda(H - F) \geq \lambda(G - F)$ .*

**Proof.** Let  $A, B$  be a partition of  $V(G)$  such that  $|E(A, B, H - F)| = \lambda(H - F)$ . If  $E(A, B, H - F) = E(A, B, G - F)$ , then we have that a min-cut in  $H - F$  is also a min-cut in  $G - F$  of the same size, thereby proving that  $\lambda(H - F) \geq \lambda(G - F)$ . Suppose not. As  $H$  is a subgraph of  $G$ ,  $E(A, B, H - F) \subseteq E(A, B, G - F)$ . Suppose  $(u, v) \in E(A, B, G - F) \setminus E(A, B, H - F)$ . Then  $(u, v) \in E(G) \setminus E(H)$ . Then from Lemma 3.2, there are  $\ell + f$  edge-disjoint paths between  $u$  and  $v$  in  $H$ , and hence there will be at least  $\ell$  edge-disjoint paths between  $u$  and  $v$  in  $H - F$ . Hence,  $\lambda(H - F) = |E(A, B, H - F)| \geq \ell = \lambda(G) \geq \lambda(G - F)$ . ◀

Proof of Theorem 1.2 follows from Lemmas 3.1, 3.2 and 3.3.

### 3.2 Lower Bound

In this section we show that the upper bound shown on  $\text{FTCS}(n, \ell, f)$  in Section 3.1 is indeed asymptotically tight. To prove Theorem 1.3, we construct an infinite family of graphs  $\mathcal{G}$ , such that for any  $G \in \mathcal{G}$  we have that any  $f$ -FTCS of  $G$  must contain all its edges. In particular, for any positive integers  $n, \ell, f$ , such that  $\frac{n-\ell-1}{\ell+f}$  is an integer, we construct a graph  $G$  on  $n$  vertices and  $(n - \ell - 1) \frac{(\ell+f-1)}{2} + (n - \ell - 1) + \frac{\ell(\ell+1)}{2}$  edges with  $\lambda(G) = \ell$ , such that any  $f$ -FTCS of  $G$  must contain all the edges of  $G$ .

Let  $n, \ell, f$  be three integers such that  $\frac{n-\ell-1}{\ell+f} = q$  is an integer. We first describe the construction of a graph  $G$  on  $n$  vertices. To easily understand our construction, we would suggest to simultaneously refer to the illustration given in Figure 1.

**Construction of a graph  $G$ .** Here,  $q = \frac{n-\ell-1}{\ell+f}$ .

- The vertex set  $V(G)$  is a union of  $X_1$  and  $X_2$ , such that  $|X_1 \cap X_2| = 1$ .
- $X_1$  has  $q$  pairwise vertex disjoint cliques  $C_1, \dots, C_q$ . Each clique  $C_i$  is on  $(\ell + f)$  vertices.  $X_1$  also contains a vertex  $a_1$  as described below. (The edges of the cliques,  $C_1, \dots, C_q$ , are denoted by the solid blue edges in Figure 1.)
- The set  $X_2$  consists of  $a_1, \dots, a_{\ell+1}$  vertices that form a clique. These vertices do not belong to the cliques,  $C_1, \dots, C_q$ . (The edges of the clique on  $a_1, \dots, a_{\ell+1}$  are represented by blue solid edges in Figure 1.)
- Let  $a_1 \in X_2$  be a fixed vertex. Each vertex in a clique  $C_i, i \in [q]$ , is adjacent to the vertex  $a_1$ . There are no edges between a pair of vertices belonging to two distinct cliques,  $C_i$  and  $C_j$ . The vertex  $a_1$  is the only common vertex between two sets  $X_1$  and  $X_2$ . (Edges between  $a_1$  and the vertices in the cliques,  $C_1, \dots, C_q$ , are represented by the red dotted edges in Figure 1.)

In the upcoming lemmas we show certain properties of our construction. Here, Lemma 3.5 is used to prove Lemma 3.6.

► **Lemma 3.4.** (★) *The number of edges in  $G$  is  $(n - \ell - 1) \frac{(\ell+f+1)}{2} + \frac{\ell(\ell+1)}{2}$ .*

► **Lemma 3.5.** (★) *For any two vertices  $u_1, u_2 \in X_1$ ,  $\lambda(u_1, u_2) \geq \ell + f$ .*

► **Lemma 3.6.** (★) *Let  $G$  be a graph and  $f \geq 1$  be a positive integer. Then  $\lambda(G) = \ell$ . Further, for any  $F \subseteq E(G[X_1])$  of size at most  $f$ , we have that  $\lambda(G - F) = \ell$ .*

We now prove the final property of an  $f$ -FTCS.

► **Lemma 3.7.** *Any  $f$ -FTCS of  $G$  must contain all the edges of  $G$ .*

**Proof.** Let  $H$  be an  $f$ -FTCS of  $G$ . We will show that  $H$  must contain all the edges of  $G$ . Towards this, we partition the edges of  $G$  into three parts, and show that all these edges are required in  $H$ . In particular, we show that if  $H$  does not include an edge of  $G$ , then there is a strategy for the adversary to choose a subset  $F$  of edges (of size at most  $f$ ) to delete from  $G$  such that  $\lambda(G - F)$  and  $\lambda(H - F)$  are not the same. Let  $u_i, i \in [\ell + f]$ , be the set of vertices of a fixed clique  $C_j$ .

- (i) Let us first show that the edges in the cliques  $C_i, i \in [q]$ , have to be present in  $H$  (the solid blue edges in  $X_1$  in Figure 1). Each  $u_i$  has  $\ell + f - 1$  edges to vertices in  $C_j$  apart from an edge to  $a_1$ . Suppose an edge  $(u_y, u_z), y, z \in [\ell + f], y \neq z$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges adjacent to  $u_z$  in  $C_j$  other than  $(u_y, u_z)$ . We know that  $f$  edges exist as  $\ell \geq 1$  (by construction  $G$  is connected). Now by Lemma 3.6 we know that  $\lambda(G - F) = \ell$ . But the degree of  $u_z$  in  $H - F$  becomes  $\ell - 1$  as  $(u_y, u_z) \notin H$ . Thus,  $\lambda(H - F) \leq \ell - 1$ . This contradicts  $H$  being an  $f$ -FTCS of  $G$ . Therefore, all edges of the cliques  $C_i$  must be present in  $H$ .
- (ii) Next, we show that edges  $E(\{a_1\}, C_i, G), i \in [q]$ , must be present in  $H$  (the red dotted edges in  $X_1$  in Figure 1). Suppose  $(u_z, a_1), z \in [\ell + f]$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges adjacent to  $u_z$  in  $C_j$  other than  $(u_z, a_1)$ . Now by Lemma 3.6 we know that  $\lambda(G - F) = \ell$ . However, the degree of  $u_z$  in  $H - F$  is  $\ell - 1$ . Thus,  $\lambda(H - F) \leq \ell - 1$ . This contradicts  $H$  being an  $f$ -FTCS of  $G$ . Therefore, for all  $i \in [q]$ , all the edges in  $E(\{a_1\}, C_i, G)$  must be present in  $H$ .
- (iii) Lastly, we show that all the edges of the  $(\ell + 1)$ -clique in  $X_2$  formed by  $a_i, i \in [\ell + 1]$  must be present in  $H$  (the solid blue edges in  $X_2$  in Figure 1). Suppose an edge

$(a_i, a_j)$ ,  $i, j \in [\ell + 1]$ ,  $i \neq j$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges of the form  $(u_i, a_1)$ ,  $i \in [f]$ . All these edges exist in  $G - F$  as  $\ell + f \geq f + 1$  (Since by construction  $G$  is connected and  $\ell \geq 1$ ). Observe that  $F \subseteq E(G[X_1])$  of size at most  $f$ , and hence by Lemma 3.6 we have that  $\lambda(G - F) = \ell$ . However,  $\lambda(H - F) = \ell - 1$ , as  $a_i$  and  $a_j$  have degree  $\ell - 1$  inside  $X_2$  in  $H - F$ . This contradicts  $H$  being an  $f$ -FTCS of  $G$ . Therefore, all the edges of the  $\ell + 1$ -clique in  $X_2$  must be present in  $H$ .

The three cases together show that if  $H$  is an  $f$ -FTCS of  $G$  then all edges of the graph  $G$  must be present in  $H$ . Thus, the total number of edges present in  $H$  is  $(n - \ell - 1) \frac{(\ell + f - 1)}{2} + (n - \ell - 1) + \frac{\ell(\ell + 1)}{2}$ . Our proof follows.  $\blacktriangleleft$

Proof of Theorem 1.3 follows from Lemmas 3.4, 3.6 and 3.7.

## 4 Min $k$ -Cut

In this section we develop upper and lower bounds on  $\text{FT-}k\text{-CS}(n, \ell, f)$ . In particular we prove Theorems 1.4 and 1.5.

### 4.1 Upper Bound

Let  $n$ ,  $\ell$  and  $f$  be three positive integers. We need to show that  $\text{FT-}k\text{-CS}(n, \ell, f)$  is upper bounded by  $\min\{(2f + \ell - (k - 1))(n - 1), (f + \ell)(n - k) + \ell\}$ . Towards this we show that given an undirected graph  $G$ , and an integer  $f \geq 1$ , we can construct an  $f$ -FT- $k$ -CS,  $H$  of  $G$  on at most  $\min\{(2f + \lambda(G, k) - (k - 1))(n - 1), ((f + \lambda(G, k))(n - k) + \lambda(G, k))\}$  edges. Indeed, when  $\lambda(G, k) = \ell$ , the upper bound follows. Our construction is presented next. It is similar to the construction of in Section 3.1 except for the choice of  $t$ .  $G$  is assumed to be connected in the algorithm. The complementary case will be handled later.

#### K-WAY-FAULT-TOLERANT-CONSTRUCTION

##### Construction of an $f$ -FT- $k$ -CS of a graph $G$ .

1. Let  $t = \min\{2f + \ell + 1 - k, f + \ell\}$ .
2. Initialize  $t$  empty (no edges) forests  $T_1, T_2, \dots, T_t$  on the same vertex set  $V(G)$ .
3. for each edge  $(u, v) \in E(G)$ , do the following.
  - Find the smallest integer  $i \in [t]$ , such that  $u$  and  $v$  are in different connected components of  $T_i$ . If no such  $i$  exists, then assign  $i$  to  $\infty$ .
  - If  $i$  is not  $\infty$  then add  $(u, v)$  to  $T_i$ .
4. Output  $H = \cup_{a=1}^t T_a$ .

Next, we show that  $H$  is an  $f$ -FT- $k$ -CS for both the values the variable  $t$  can take. We start with the following observation which we use in both the cases.

► **Lemma 4.1.** ( $\star$ ) *Let  $(u, v) \in E(G) \setminus E(H)$ . Then there are at least  $t$  edge-disjoint paths between  $u$  and  $v$  in  $G$  and  $H$ .*

Note that the  $t(n - 1)$  upper bound of Section 3.1 for the number of edges in  $H$  applies here too with the same proof. However, we show stronger bounds for certain values of  $t$ .

### 4.1.1 Case of $t = f + \ell$

► **Lemma 4.2.** *The subgraph  $H$  has at most  $(f + \ell)(n - k) + \ell$  edges.*

**Proof.** Let  $A_1, \dots, A_k$  be a partition of  $V(G)$  such that  $|E(A_1, \dots, A_k, G)| = \ell$ . Let  $X = E(A_1, \dots, A_k, G)$ . We will show that every forest  $T_i - X$ ,  $i \in [f + \ell]$ , has at least  $k$  connected components. Note that, once we can show this claim, we can get the upper bound on the number of edges in  $H$ . Indeed, each  $T_i - X$  has at most  $n - k$  edges (since, it has at least  $k$  components) and hence  $|E(H)| \leq \sum_{i=1}^{f+\ell} |E(T_i - X)| + |X| \leq (f + \ell)(n - k) + \ell$ . Next we prove our claim. Observe that every edge going out of the connected components  $A_j$ ,  $j \in [k]$ , is contained inside  $X$ . Thus, in particular, every edge going out of the vertices in  $A_j$  in  $T_i$  is also contained inside  $X$ . Hence, the vertices of  $A_j$  at least form one connected component in  $T_i - X$ . This concludes the proof that every forest  $T_i - X$ ,  $i \in [f + \ell]$ , has at least  $k$  connected components. ◀

Next, we show that  $H$  is an  $f$ -FT- $k$ -CS.

► **Lemma 4.3.** *Let  $G$  be a graph with  $\lambda(G, k) = \ell$ ,  $f$  be a positive integer, and  $H$  be the subgraph constructed above. Then, for any set  $F$  of at most  $f$  edges,  $\lambda(H - F, k) \geq \lambda(G - F, k)$ .*

**Proof.** Let  $A_1, \dots, A_k$  be a partition of  $V(G)$  such that  $|E(A_1, \dots, A_k, H - F)| = \lambda(H - F, k)$ . If  $E(A_1, \dots, A_k, H - F) = E(A_1, \dots, A_k, G - F)$ , then we have that a  $k$ -cut in  $H - F$  is also a  $k$ -cut in  $G - F$  of the same size, thereby proving that  $\lambda(H - F, k) \geq \lambda(G - F, k)$ . Suppose not. As  $H$  is a subgraph of  $G$ ,  $E(A_1, \dots, A_k, H - F) \subseteq E(A_1, \dots, A_k, G - F)$ . Suppose  $(u, v) \in E(A_1, \dots, A_k, G - F) \setminus E(A_1, \dots, A_k, H - F)$ . Then  $(u, v) \in E(G) \setminus E(H)$ . Then from Lemma 4.1, there are  $\ell + f$  edge-disjoint paths between  $u$  and  $v$  in  $H$ , and hence there will be at least  $\ell$  edge-disjoint paths between  $u$  and  $v$  in  $H - F$ . Hence,  $\lambda(H - F, k) = |E(A_1, \dots, A_k, H - F)| \geq \ell = \lambda(G, k) \geq \lambda(G - F, k)$ . This concludes the proof. ◀

### 4.1.2 Case of $t = 2f + \ell + 1 - k$

We will show that  $H$  is an  $f$ -FT- $k$ -CS with at most  $(2f + \ell + 1 - k)(n - 1)$  edges.

The bound on the number of edges on  $H$  is clear, as  $H$  is the union of at most  $(2f + \ell + 1 - k)$  forests.

► **Lemma 4.4.** *The subgraph  $H$  has at most  $(2f + \ell + 1 - k)(n - 1)$  edges.*

We could have obtained a bound similar to Lemma 4.2, but in this case, it does not give us asymptotically better bound than that of  $(2f + \ell + 1 - k)(n - 1)$ . Next, we show that  $H$  is an  $f$ -FT- $k$ -CS. We start with the following lemma which is a folklore and we give the proof here for completeness.

► **Lemma 4.5.**  $(\star)$  *Let  $G$  be a connected graph and let  $u_1, \dots, u_p \in V(G)$ . Further, let  $E_{[p]}$  be an inclusion-wise minimal subset of edges, such that  $u_1, \dots, u_p$  get pairwise separated in  $G - E_{[p]}$ , then  $G - E_{[p]}$  has exactly  $p$  connected components, one containing each  $u_i$ ,  $i \in [p]$ .*

► **Lemma 4.6.** *Let  $G$  be a connected graph, then for all  $p \leq k$ , we have that  $\lambda(G, k) \geq \lambda(G, p) + (k - p)$ .*

**Proof.** Let  $A_1, \dots, A_k$  be a  $k$  partition of  $V(G)$  such that  $|E(A_1, \dots, A_k, G)| = \lambda(G, k)$ . Let  $u_i \in A_i$ ,  $i \in [p]$ , be a vertex. Clearly,  $E(A_1, \dots, A_k, G)$  separates any pair of vertices in  $\{u_1, \dots, u_p\}$ , and thus there exists an inclusion-wise minimal subset  $E_{[p]} \subseteq E(A_1, \dots, A_k, G)$ ,

such that any pair of vertices in  $\{u_1, \dots, u_p\}$  gets separated in  $G - E_{[p]}$ . Now by Lemma 4.5, we have that  $G - E_{[p]}$  has exactly  $p$  connected components, one containing each  $u_i$ ,  $i \in [p]$ . This implies that  $E_p$  is a  $p$ -cut in  $G$  (may not be of the minimum size) and thus,  $|E_{[p]}| \geq \lambda(G, p)$ .

However,  $G - E(A_1, \dots, A_k, G)$  has  $k$  connected components, and deleting an edge can only increase the number of connected components by 1. This implies that  $|E(A_1, \dots, A_k, G) \setminus E_{[p]}| \geq (k - p)$ . Putting together this with the fact that  $|E_{[p]}| \geq \lambda(G, p)$ , we get that

$$\lambda(G, k) \geq |E_{[p]}| + (k - p) \geq \lambda(G, p) + (k - p).$$

This concludes the proof. ◀

To prove that  $H$  is an  $f$ -FT- $k$ -CS of  $G$ , we need to show that for any set of edges  $F \subseteq E(G)$  of cardinality at most  $f$ ,  $\lambda(H - F, k) = \lambda(G - F, k)$ . As  $H$  is a subgraph of  $G$ , we know from Lemma 2.2 that  $\lambda(H - F, k) \leq \lambda(G - F, k)$ . Now we show that  $\lambda(H - F, k) \geq \lambda(G - F, k)$ . In fact, we will prove something stronger, which we call *robustness*. That is, for all  $k^* \leq k$ , we have that  $\lambda(H - F, k^*) \geq \lambda(G - F, k^*)$ .

► **Lemma 4.7 (Robustness).** *Let  $G$  be a connected graph with  $\lambda(G, k) = \ell$ ,  $f$  be a positive integer, and  $H$  be the subgraph constructed above. Then, for any set  $F$  of at most  $f$  edges, and for  $k^* \leq k$ ,  $\lambda(H - F, k^*) \geq \lambda(G - F, k^*)$ .*

**Proof.** Let  $A_1, \dots, A_{k^*}$  be a partition into  $k^*$  sets of  $V(G)$  such that  $|E(A_1, \dots, A_{k^*}, H - F)| = \lambda(H - F, k^*)$ . If  $E(A_1, \dots, A_{k^*}, H - F) = E(A_1, \dots, A_{k^*}, G - F)$ , then we have that a min  $k^*$ -cut in  $H - F$  is also a min  $k^*$ -cut in  $G - F$  of the same size, thereby proving that  $\lambda(H - F, k^*) \geq \lambda(G - F, k^*)$ . Suppose not. As  $H$  is a subgraph of  $G$ ,  $E(A_1, \dots, A_{k^*}, H - F) \subseteq E(A_1, \dots, A_{k^*}, G - F)$ . Suppose  $(u, v) \in E(A_i, A_j, G - F) \setminus E(A_i, A_j, H - F)$ ,  $i, j \in [k^*]$ , and  $i \neq j$ . Then  $(u, v) \in E(G) \setminus E(H)$ . From Lemma 4.1, there are  $2f + \ell + 1 - k$  edge-disjoint paths between  $u$  and  $v$  in  $H$ , and hence there will be at least  $f + \ell + 1 - k$  edge-disjoint paths between  $u$  and  $v$  in  $H - F$ .

Observe that, since  $G$  is connected,  $H$  is also connected by our construction ( $T_1$  is definitely a spanning tree). However,  $H - F$  may not be *connected*. On the other hand, since  $\lambda_H(u, v) \geq 2f + \ell + 1 - k$ , we get that  $\lambda_{H-F}(u, v) \geq f + \ell + 1 - k$ . Note that since,  $H$  is connected, any  $k$ -cut has size at least  $k - 1$ , and thus,  $\ell + 1 \geq k$  (recall that,  $\lambda(G, k) = \ell$ ). Since,  $\lambda_{H-F}(u, v) \geq f + \ell + 1 - k \geq f \geq 1$ , we have that  $u$  and  $v$  are in the same connected component of  $H - F$ . Further, they get separated after we delete  $E(A_1, \dots, A_{k^*}, H - F)$  from  $H - F$ . This implies that the number of connected components in  $H - F$  is at most  $k^* - 1$ . Next observe that since  $H$  is connected, deleting  $F$  from  $H$  can only result in at most  $|F| + 1$  connected components in  $H - F$ . Thus, the number of connected components in  $H - F$ , say  $d$ , is upper bounded by the minimum of  $\{k^* - 1, f + 1\}$ .

Let the connected component containing  $u, v$  in  $H - F$  be denoted by  $C_{uv}$ . Observe that  $E(A_1, \dots, A_{k^*}, H - F)$  separates  $u$  from  $v$  in  $H - F - E(A_1, \dots, A_{k^*}, H - F)$ , and thus there exists an inclusion-wise minimal subset  $E_{uv} \subseteq E(A_1, \dots, A_{k^*}, H - F)$ , such that  $u$  and  $v$  get separated in  $(H - F) - E_{uv}$ . Further, note that the minimality of  $E_{uv}$  implies that  $E_{uv} \subseteq E(C_{uv})$ , and it is an inclusion-wise minimal separator for  $u$  and  $v$  in  $C_{uv}$ . Applying Lemma 4.5 on  $C_{uv}$ , we get that  $C_{uv} - E_{uv}$  has exactly two connected components,  $C_u$  and  $C_v$ , containing  $u, v$ , respectively. This implies that  $|E_{uv}| \geq \lambda_{H-F}(u, v) = \lambda_{C_{uv}}(u, v) \geq f + \ell + 1 - k$ . Recall that,  $H - F$  has  $d$  components, and thus  $H - F - E_{uv}$  has  $d + 1$  components. However,  $H - F - E(A_1, \dots, A_{k^*}, H - F)$  has  $k^*$  connected components. This implies that

## 10:12 Optimal Output Sensitive Fault Tolerant Cuts

$|E(A_1, \dots, A_{k^*}, H - F) \setminus E_{uv}| \geq (k^* - d)$ . Hence,

$$\begin{aligned}
\lambda(H - F, k^*) &= |E(A_1, \dots, A_{k^*}, H - F)| \\
&\geq f + \ell + 1 - k + (k^* - d) \\
&\geq \ell + (f + 1) - (f + 1) - (k - k^*) && \text{(Using } d \leq f + 1\text{)} \\
&= \ell - (k - k^*) \\
&= \lambda(G, k) - (k - k^*) && \text{(since, } \lambda(G, k) = \ell\text{)} \\
&\geq \lambda(G - F, k^*) + (k - k^*) - (k - k^*) && \text{(Lemma 4.6)} \\
&= \lambda(G - F, k^*).
\end{aligned}$$

This concludes the proof.  $\blacktriangleleft$

Now we deal with the case when  $G$  is not connected.

► **Lemma 4.8.** *Let  $G$  be a disconnected graph with  $d > 1$  connected components with  $\lambda(G, k) = \ell$  and let  $f$  be a positive integer. Then there exists a subgraph  $H$  of  $G$  with at most  $(n - d)(2f + \ell + 2 - k + d)$  edges such that for any set  $F$  of at most  $f$  edges,  $\lambda(H - F, k) \geq \lambda(G - F, k)$ .*

**Proof.** If  $d \geq k$ , then we return  $H$  as an empty (edgeless) graph on the vertices of  $G$ . So let us assume that  $d < k$ . Suppose,  $G$  has connected components  $G_1, \dots, G_d$ . We apply K-WAY-FAULT-TOLERANT-CONSTRUCTION with  $G_i, i \in [d]$  and  $k' = (k - d + 1)$  and get  $H_i$ . Let  $H = \cup_{i=1}^d H_i$ . That is, we apply our upper bound construction on *each of the connected components* with  $k'$  and get the desired  $H$ . Lemma 4.7 implies the following.

▷ **Claim 4.9.** For all  $i \in [d]$ ,  $k^* \leq k'$ ,  $H_i$  is a  $f$ -FT- $k^*$ -CS of  $G_i$ .

Next we show that for any set  $F$  of at most  $f$  edges,  $\lambda(H - F, k) \geq \lambda(G - F, k)$ . Let  $A_1, \dots, A_k$  be a  $k$  partition of  $V(G)$  such that  $|E(A_1, \dots, A_k, H - F)| = \lambda(H - F, k)$ . Observe that, since  $G_i$  is connected we have that  $H_i$  is connected. Let  $A_1, \dots, A_k$  be a  $k$  partition of  $V(G)$  such that  $|E(A_1, \dots, A_k, H - F)| = \lambda(H - F, k)$ . Recall, that  $d < k$ , thus,  $\lambda(H - F, k) > 0$ . Further, from the minimality of  $E(A_1, \dots, A_k, H - F)$ , and the fact that  $|E(A_1, \dots, A_k, H - F)| \geq 1$ , we have that  $H[A_i]$  is connected, and completely contained inside one of  $G_j, j \in [d]$ . That is,  $E(A_1, \dots, A_k, H - F)$  further partitions some of the connected components, and each connected component of  $H - E(A_1, \dots, A_k, H - F)$  is a part in the partition  $(A_1, \dots, A_k)$  of  $V(G)$ .

Let  $E_i = E(H_i) \cap E(A_1, \dots, A_k, H - F)$ ,  $i \in [d]$  and  $\ell_i = |E_i|$ . Further, let  $\mathcal{A}_i, i \in [d]$ , be the set of parts among  $A_1, \dots, A_k$ , which are completely contained inside  $G_i$ . Note that for all  $i, j \in [d]$ ,  $i \neq j$ ,  $\mathcal{A}_i$  and  $\mathcal{A}_j$  are pairwise disjoint, and for all  $q \in [k]$ , there exists an integer  $j \in [d]$ , such that  $A_q \in \mathcal{A}_j$ . We summarize this in the following claim.

▷ **Claim 4.10.** Let  $k_i = |\mathcal{A}_i|$ , then  $\sum_{i=1}^d k_i = k$ .

Next we have the following claim.

▷ **Claim 4.11.** For all  $i \in [d]$ ,  $k_i \leq k'$ ,  $\lambda(H_i - F, k_i) = \ell_i$ . Further,  $\sum_{i=1}^d \ell_i = \ell$ .

**Proof.** No  $k_i$  can be more than  $k'$ , otherwise we get strictly greater than  $k$  components, contradicting that  $H - F - E(A_1, \dots, A_k, H - F)$  has exactly  $k$  components. Further, for some  $i$ , let  $\lambda(H_i, k_i) = \ell'_i < \ell_i$ . In this case, we can delete  $\ell'_i$  edges inside  $G_i$ , and delete  $E_j, j \neq i$ , to get  $k$  components in  $H - F$ , by deleting strictly less than  $\ell$  edges. This contradicts the definition of  $E(A_1, \dots, A_k, H - F)$ . By definition  $\sum_{i=1}^d \ell_i = \ell$ .  $\blacktriangleleft$

$$\begin{aligned}
\lambda(H - F, k) &= \sum_{i=1}^d \lambda(H_i - F, k_i) && \text{(Claim 4.11)} \\
&\geq \sum_{i=1}^d \lambda(G_i - F, k_i) && \text{(Claim 4.9)} \\
&= \lambda(G - F, k).
\end{aligned}$$

To see the last inequality observe the following. Let  $W_i$  be a subset of edges in  $E(G_i)$  such that  $|W_i| = \lambda(G_i - F, k_i)$ . Then, clearly by deleting  $W = \cup_{i=1}^d W_i$ , we get  $\sum_{i=1}^d k_i = k$  components in  $G - F$ . Here, we used Claim 4.10 to conclude that  $\sum_{i=1}^d k_i = k$ . This implies that  $\sum_{i=1}^d \lambda(G_i - F, k_i) = \sum_{i=1}^d |W_i|$  is a  $k$ -cut of  $G - F$  (may not be of the minimum size). Thus, a min  $k$ -cut in  $G - F$  can only be *smaller*. This concludes the correctness proof.

All that remains to show is the upper bound on the number of edges. Let the number of vertices in each component be  $n_i, i \in [d]$ . Then, the total number of edges in  $H$  is upper bounded as follows.

$$|E(H)| \leq \sum_{i=1}^d (n_i - 1)(2f + \ell + 1 - k') = (n - d)(2f + \ell + 1 - k') = (n - d)(2f + \ell + 2 - k + d).$$

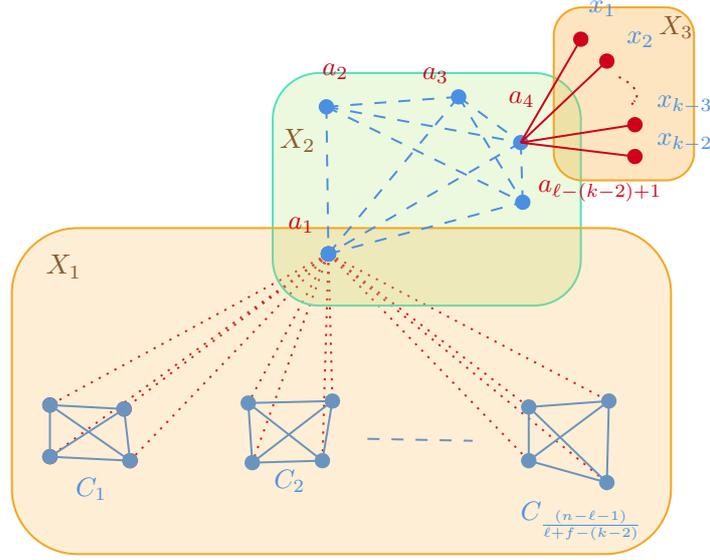
This concludes the proof. ◀

Proof of Theorem 1.4 follows from Lemmas 4.2, 4.3, 4.4, 4.7 and 4.8.

## 4.2 Lower Bound

In this section we show that the upper bound shown on FT- $k$ -CS( $n, \ell, f$ ) in Section 4.1 is indeed asymptotically tight. To prove Theorem 1.5, we construct an infinite family of graphs  $\mathcal{G}$ , such that for any  $G \in \mathcal{G}$  we have that any  $f$ -FT- $k$ -CS of  $G$  must contain all its edges. In particular, for any positive integers  $n, \ell, f$ , such that  $\frac{n-\ell-1}{\ell+f-(k-2)}$  is an *integer*, we construct a graph  $G$  on  $n$  vertices and  $\frac{(n-\ell-1)(\ell+f-k+1)}{2} + n - \ell + k - 3 + \frac{(\ell-k+3)(\ell-k+2)}{2}$  edges with  $\lambda(G, k) = \ell$ , such that any  $f$ -FT- $k$ -CS of  $G$  must contain all the edges of  $G$ .

The graph  $G$  is a modification of the graph used to show the lower bound for global minimum cut in Section 3.2.



■ **Figure 2** Vertex  $a_1$  has degree  $n - \ell - 1$  within  $X_1$ . In  $X_2$  the vertices  $a_1, a_2, \dots, a_{\ell-(k-2)+1}$  induce a clique. The vertex  $a_4$  has  $k - 2$  edges in  $X_3$  each going to a separate vertex  $x_i$ . There are no edges between  $x_i$ 's.  $C_1, C_2, \dots, C_{\frac{(n-\ell-1)}{\ell+f-(k-2)}}$  represent  $\frac{(n-\ell-1)}{\ell+f-(k-2)}$  cliques each of size  $\ell + f - (k - 2)$  within  $X_1$ .

**Construction of a graph  $G$ .** Here,  $q = \frac{n-\ell-1}{\ell+f-(k-2)}$ .

- The vertex set  $V(G)$  is a union of  $X_1, X_2$  and  $X_3$ , such that  $|X_1 \cap X_2| = 1$ .
- $X_1$  has  $q$  pairwise vertex disjoint cliques  $C_1, \dots, C_q$ . Each clique  $C_i$  is on  $(\ell + f - (k - 2))$  vertices. (The edges of the cliques,  $C_1, \dots, C_q$  are denoted by the solid blue edges in Figure 2.)
- The set  $X_2$  consists of  $a_1, \dots, a_{\ell-(k-2)+1}$  vertices that form a clique. These vertices do not belong to the cliques,  $C_1, \dots, C_q$ . (The edges of the clique on  $a_1, \dots, a_{\ell-(k-2)+1}$  are represented by blue solid edges in  $X_2$  in Figure 2.)
- Let  $a_1 \in X_2$  be a fixed vertex. All the vertices in a clique  $C_i, i \in [q]$ , is adjacent to the vertex  $a_1$ . There are no edges between a pair of vertices belonging to two distinct cliques,  $C_i$  and  $C_j$ . The vertex  $a_1$  is the only common vertex between two sets  $X_1$  and  $X_2$ . (Edges between  $a_1$  and the vertices in the cliques,  $C_1, \dots, C_q$ , are represented by the red dotted edges in Figure 2.)
- $X_3$  consists of  $k - 2$  vertices  $x_i, i \in [k - 2]$ . Let  $a_4 \in X_2$  be a fixed vertex. Edges in  $X_3$  are of the form  $(a_4, x_i), i \in [k - 2]$ . There are no edges between  $x_i$ 's. (Edges between  $a_4$  and the vertices  $x_i \in X_3$ , are represented by the red solid edges in Figure 2.)

In the upcoming lemmas we show certain properties of our construction.

► **Lemma 4.12.** *The number of edges in  $G$  is  $\frac{(n-\ell-1)(\ell+f-k+1)}{2} + n - \ell + k - 3 + \frac{(\ell-k+3)(\ell-k+2)}{2}$ .*

**Proof.** Each clique  $C_i$  is of size  $(\ell + f - (k - 2))$  and contributes  $(\ell + f - (k - 2)) \binom{\ell+f-k+1}{2}$  edges. There are  $q$  cliques and thus the total number of edges contributed by all cliques  $C_i, i \in [q]$  is  $(n - \ell - 1) \binom{\ell+f-k+1}{2}$ . The vertex  $a_1$  is adjacent to all vertices of all  $C_i$ 's. Hence,  $a_1$  has degree  $n - \ell - 1$  inside  $X_1$ . The  $(\ell - (k - 2) + 1)$ -clique in  $X_2$  contributes  $\frac{(\ell-k+3)(\ell-k+2)}{2}$  edges. The vertex  $a_4$  contributes  $k - 2$  edges in  $X_3$ . Therefore, the total number of edges of  $G$  is  $\frac{(n-\ell-1)(\ell+f-k+1)}{2} + n - \ell + k - 3 + \frac{(\ell-k+3)(\ell-k+2)}{2}$ . ◀

► **Lemma 4.13.** *For any two vertices  $u_1, u_2 \in X_1$ ,  $\lambda(u_1, u_2) \geq \ell + f - k + 2$ .*

**Proof.** The pair  $\{u_1, u_2\}$  is of one of the three types described below. We prove the claim for each of the three types.

- Both  $u_1$  and  $u_2$  are part of the same clique  $C_i$  in  $X_1$ . We know that the size of  $C_i$  is  $\ell + f - k + 2$ . Let the other vertices in  $C_i$  be  $u_j, j \in [3, \ell + f - k + 2]$ . Then  $u_1 u_j u_2, j \in [3, \ell + f - k + 2], u_1, u_2$  and  $u_1 a_1 u_2$  are  $\ell + f - k + 2$  edge-disjoint paths between  $u_1$  and  $u_2$ . By Theorem 2.1  $\lambda(u_1, u_2) \geq \ell + f - k + 2$ .
- Vertices  $u_1 \in C_i$  and  $u_2 \in C_j$ , and  $i \neq j$ . Let  $v_j, j \in [\ell + f - k + 1]$  denote the vertices in  $C_j$  other than  $u_2$ . Let  $w_1$  be a vertex in  $C_i$  other than  $u_1$ . Then  $u_1 a_1 v_j u_2, j \in [\ell + f - k + 1]$  and  $u_1 w_1 a_1 u_2$  are  $\ell + f - k + 2$  edge-disjoint paths between  $u_1$  and  $u_2$ . By Theorem 2.1  $\lambda(u_1, u_2) \geq \ell + f - k + 2$ .
- Let  $u_1$  be a part of clique  $C_i$  and  $u_2 = a_1$ . Let  $v_j, j \in [\ell + f - k + 1]$  denote the vertices in  $C_i$  other than  $u_1$ . Then  $u_1 v_j a_1, j \in [\ell + f - k + 1]$  and  $u_1 a_1$  are  $\ell + f - k + 2$  edge-disjoint paths between  $u_1$  and  $a_1$ . By Theorem 2.1 that  $\lambda(u_1, u_2) \geq \ell + f - k + 2$ .

This concludes the proof. ◀

► **Lemma 4.14.** *Let  $G$  be a graph and  $f \geq 1$  be a positive integer. Then  $\lambda(G, k) = \ell$ . Further, for any  $F \subseteq E(G[X_1])$  of size at most  $f$ , we have that  $\lambda(G - F, k) = \ell$ .*

**Proof.** Vertices  $x_i, i \in [k - 2]$  in  $X_3$  have degree 1 with all of them adjacent to  $a_4$ . The edges  $E(x, X_1 \cup X_2, G)$  for all  $x = x_i, i \in [k - 2]$  partition the graph into  $k - 1$  components using  $k - 2$  edges. As a minimum of  $k - 2$  edges are required to partition a connected graph into  $k - 1$  components all these edges will be part of  $\lambda(G, k)$ . We need one more partition of the graph to get  $k$  components.

The cut  $E(\{a\}, X_1 \cup X_2 \setminus \{a\}, G)$ , where  $a = a_i$  for  $i \in [\ell - k + 3]$  is of size  $\ell - k + 2$ . Together, with the edges  $E(x, X_1 \cup X_2, G)$  for all  $x = x_i, i \in [k - 2]$  we get a  $k$ -cut of  $G$  of size  $\ell$ .

Now we show that any other cut if of size at least  $\ell$ . From Lemma 4.13 we know that for any two vertices  $u_1, u_2 \in X_1$ ,  $\lambda(u_1, u_2) \geq \ell + f - k + 2$ . This implies that for any 2 partitions  $A, B$  of  $V(G)$  such that  $|X_1 \cap A| \geq 1$  and  $|X_1 \cap B| \geq 1$  we have that  $|E(A, B, G)| \geq \ell + f - k + 2 \geq \ell - k + 3$ . In this case,  $\lambda(G - F, k) \geq \ell - k + 3 + (k - 2) = \ell + 1$ . Thus, any min- $k$ -cut should keep all of  $X_1$  in one side of the partition. It can be easily checked that  $|E(X_1 \cup Y, X_2 \setminus Y, G)| \geq \ell - k + 2$  for any  $Y \subseteq X_2$ , with the minimum being achieved when  $Y$  is a singleton set. These edges along with  $k - 2$  edges from  $X_3$  shows that  $\lambda(G, k) = \ell$ . This concludes the first part of the proof.

Let  $F \subseteq E(G[X_1])$  of size at most  $f$  and  $A_i, i \in [k]$  be a partitioning of  $V(G)$ . We will show that  $|E(A_1, \dots, A_k, G - F)| \geq \ell$ . Indeed, if  $|X_1 \cap A_i| \geq 1$  and  $|X_1 \cap A_j| \geq 1$  for  $i \neq j$ , we have that  $|E(A_i, A_j, G - F)| \geq \ell - k + 2$  (since,  $|E(A_i, A_j, G)| \geq \ell + f - k + 2$ ). These edges alongwith the  $k - 2$  edges  $(a_4, x_i), i \in [k - 2]$  give  $|E(A_1, \dots, A_k, G - F)| \geq \ell$ . Thus, let us assume that all of  $X_1$  in one side of the partition. Again in this case, we can easily check that  $|E(X_1 \cup Y, X_2 \setminus Y, G - F)| \geq \ell - k + 2$  for any  $Y \subseteq X_2$ , with the minimum being achieved when  $Y$  is a singleton set. Alongwith the edges  $(a_4, x_i), i \in [k - 2]$ , we have that  $|E(A_1, \dots, A_k, G - F)| \geq \ell$ . Thus,  $\lambda(G - F, k) = \ell$ . This concludes the proof. ◀

We now prove the final property of an  $f$ -FT- $k$ -CS.

► **Lemma 4.15.** *Any  $f$ -FT- $k$ -CS of  $G$  must contain all the edges of  $G$ .*

**Proof.** Let  $H$  be an  $f$ -FT- $k$ -CS of  $G$ . We will show that  $H$  must contain all edges of  $G$ . Towards this, we partition the edges of  $G$  into four parts, and show that all these edges are required in  $H$ . In particular, we show that if  $H$  does not include an edge of  $G$ , then there is

a strategy for the adversary to choose a subset  $F$  of edges (of size at most  $f$ ) to delete from  $G$  such that  $\lambda(G - F, k)$  and  $\lambda(H - F, k)$  are not the same. Let  $u_i, i \in [\ell + f - k + 2]$ , be the set of vertices of a fixed clique  $C_j$ .

(i) We first show that the edges in the cliques  $C_i, i \in [q]$  in  $X_1$  are present in  $H$  (the solid blue edges in  $X_1$  in Figure 2). Each  $u_i$  has degree  $\ell + f - k + 1$  within  $C_j$  apart from an edge to  $a_i$ . Suppose an edge  $(u_y, u_z), y, z \in [\ell + f - k + 2], y \neq z$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges adjacent to  $u_z$  in  $C_j$  other than  $(u_y, u_z)$ . We know that  $f$  edges exist as  $\ell \geq k - 1$  (by construction  $G$  is connected).

Now by Lemma 4.14 we know that  $\lambda(G - F, k) = \ell$ . But the degree of  $u_z$  in  $H - F$  becomes  $\ell - k + 1$  as  $(u_y, u_z) \notin H - F$ . In  $H - F$ , we will choose all the remaining adjacent edges of  $u_z$  and the  $k - 2$  edges in  $X_3$  as our cut edges. Thus,  $\lambda(H - F, k) = \ell - 1$ . This contradicts  $H$  being an  $f$ -FT- $k$ -CS of  $G$ . Therefore, all edges of the cliques  $C_i$  must be present in  $H$ .

(ii) Next, we show that the edges  $E(\{a_1\}, C_i, G)$  are present in  $H$  (the red dotted edges in  $X_1$  in Figure 2). Suppose  $(u_z, a_1), z \in [\ell + f - k + 2]$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges adjacent to  $u_z$  in  $C_j$  other than  $(u_z, a_1)$ . By Lemma 4.14, we know that  $\lambda(G - F) = \ell$  but  $\lambda(H - F) = \ell - 1$ . A similar argument to case (i), shows that all such edges  $E(\{a_1\}, C_i, G)$  must be present in  $H$ .

(iii) Next, let us show that the edges in the  $\ell - k + 3$ -clique in  $X_2$  formed by  $a_i, i \in [\ell - k + 3]$  are present in  $H$  (the dashed blue edges in  $X_2$  in Figure 2). Suppose edge  $(a_i, a_j), i, j \in [\ell - k + 3], i \neq j$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges of the form  $(u_i, a_1), i \in [f]$ . All these edges exist in  $G - F$  as  $\ell + f - k + 2 \geq f + 1$  (Since  $G$  is connected and  $\ell \geq k - 1$ ).

By Lemma 4.14 we have that  $\lambda(G, k) = \ell$ . However, as  $a_i$  and  $a_j$  both have degree  $\ell - k + 1$  inside  $X_2$  in  $H$  so  $E(\{a_i\}, X_1 \cup X_2 \setminus \{a_i\}, H - F)$  or  $E(\{a_j\}, X_1 \cup X_2 \setminus \{a_j\}, H - F)$  alongwith  $k - 2$  edges in  $X_3$  give  $\lambda(H - F, k) = \ell - 1$ . This contradicts  $H$  being an  $f$ -FT- $k$ -CS of  $G$ . Therefore, all edges of the  $\ell - k + 3$ -clique in  $X_2$  must be present in  $H$ .

(iv) Lastly, we show that all the  $k - 2$  edges  $E(x, X_1 \cup X_2, G)$  for all  $x = x_i, i \in [k - 2]$  are present in  $H$  (the solid red edges in  $X_3$  in Figure 2). Suppose an edge  $(a_4, x_z), z \in [k - 2]$  is not present in  $H$ . Let  $F$  consist of any  $f$  edges of the form  $(u_i, a_1), i \in [f]$ .

Again by Lemma 4.14 we have that  $\lambda(G - F, k) = \ell$ . However, as edge  $(a_4, x_z) \notin H$ , we have that  $\lambda(H - F, k) = \ell - 1$ . This contradicts  $H$  being an  $f$ -FT- $k$ -CS of  $G$ . Therefore, all  $k - 2$  edges of  $X_3$  must be present in  $H$ .

All the cases together show that all edges of the graph  $G$  must be present in  $H$  if  $H$  is an  $f$ -FT- $k$ -CS of  $G$ . Thus, the total number of edges present in  $H$  is  $\frac{(n - \ell - 1)(\ell + f - k + 1)}{2} + n - \ell + k - 3 + \frac{(\ell - k + 3)(\ell - k + 2)}{2}$ . Our proof follows. ◀

Proof of Theorem 1.5 follows from Lemmas 4.12, 4.14 and 4.15.

---

## References

- 1 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7\_4.
- 2 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11*

- August 2017, volume 70 of *Proceedings of Machine Learning Research*, pages 311–321. PMLR, 2017. URL: <http://proceedings.mlr.press/v70/backurs17a.html>.
- 3 Surender Baswana, Shreejit Ray Choudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: breaking the  $O(m)$  barrier. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 730–739, 2016.
  - 4 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant reachability for directed graphs. In *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 528–543, 2015.
  - 5 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 167–178, 2015.
  - 6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.
  - 7 Gilad Braunschvig, Shiri Chechik, David Peleg, and Adam Sealfon. Fault tolerant additive and  $(\mu, \alpha)$ -spanners. *Theor. Comput. Sci.*, 580:94–100, 2015.
  - 8 Diptarka Chakraborty and Keerti Choudhary. New extremal bounds for reachability and strong-connectivity preservers under failures. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 25:1–25:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.25.
  - 9 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010.
  - 10 Chandra Chekuri, Kent Quanrud, and Chao Xu. LP relaxation and tree packing for minimum  $k$ -cuts. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 7:1–7:18, 2019. doi:10.4230/OASICS.SOSA.2019.7.
  - 11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
  - 12 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 169–178, 2011.
  - 13 Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto-Rodriguez, and Frances A. Rosamond. Cutting up is hard to do: the parameterized complexity of  $k$ -cut and related problems. *Electron. Notes Theor. Comput. Sci.*, 78:209–222, 2003. doi:10.1016/S1571-0661(04)81014-4.
  - 14 Shimon Even. An algorithm for determining whether the connectivity of a graph is at least  $k$ . *SIAM J. Comput.*, 4(3):393–396, 1975. doi:10.1137/0204034.
  - 15 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 112–122, 1991. doi:10.1145/103418.103436.
  - 16 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in  $O(m \log^2 n)$  time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.57.
  - 17 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1260–1279. SIAM, 2020. doi:10.1137/1.9781611975994.77.

- 18 Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the  $k$ -cut problem for fixed  $k$ . *Math. Oper. Res.*, 19(1):24–37, 1994. doi:10.1287/moor.19.1.24.
- 19 Anupam Gupta, Euiwoong Lee, and Jason Li. The karger-stein algorithm is optimal for  $k$ -cut. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 473–484, 2020. doi:10.1145/3357713.3384285.
- 20 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 127:1–127:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.127.
- 21 Nili Guttman-Beck and Refael Hassin. Approximation algorithms for minimum  $K$ -cut. *Algorithmica*, 27(2):198–207, 2000. doi:10.1007/s004530010013.
- 22 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1919–1938, 2017. doi:10.1137/1.9781611974782.125.
- 23 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM J. Comput.*, 49(1):1–36, 2020. doi:10.1137/18M1180335.
- 24 David R. Karger. Global min-cuts in  $\text{rnc}$ , and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30, 1993. URL: <http://dl.acm.org/citation.cfm?id=313559.313605>.
- 25 David R. Karger. Using randomized sparsification to approximate minimum cuts. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 424–432. ACM/SIAM, 1994. URL: <http://dl.acm.org/citation.cfm?id=314464.314582>.
- 26 David R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 424–432, 1994.
- 27 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.
- 28 David R. Karger and Clifford Stein. An  $\tilde{O}(n^2)$  algorithm for minimum cuts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 757–765, 1993. doi:10.1145/167088.167281.
- 29 David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. doi:10.1145/234533.234534.
- 30 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 665–674, 2015. doi:10.1145/2746539.2746588.
- 31 Daniel Lokshantov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min  $k$ -cut. *CoRR*, to appear in *FOCS 2020*, abs/2005.00134, 2020. arXiv:2005.00134.
- 32 David W. Matula. A linear time  $2+\epsilon$  approximation algorithm for edge connectivity. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 500–504, 1993. URL: <http://dl.acm.org/citation.cfm?id=313559.313872>.
- 33 Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.* 10:, pages 96–115, 1927.
- 34 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT*

- Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.
- 35 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse  $k$ -connected spanning subgraph of a  $k$ -connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.
  - 36 Merav Parter. Vertex fault tolerant additive spanners. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 167–181, 2014.
  - 37 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490, 2015.
  - 38 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.
  - 39 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1073–1092, 2014.
  - 40 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.
  - 41 Mikkel Thorup. Minimum  $k$ -way cuts via deterministic greedy tree packing. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 159–166. ACM, 2008. doi:10.1145/1374376.1374402.
  - 42 Vijay Vazirani. Approximation algorithms. *Berlin: Springer, ISBN 978-3-540-65367-7*, 2003.



# Online Matching with Recourse: Random Edge Arrivals

**Aaron Bernstein**

Rutgers University, New Brunswick, NJ, USA  
bernstei@gmail.com

**Aditi Dudeja**

Rutgers University, New Brunswick, NJ, USA  
aditi.dudeja@rutgers.edu

---

## Abstract

The matching problem in the online setting models the following situation: we are given a set of servers in advance, the clients arrive one at a time, and each client has edges to some of the servers. Each client must be matched to some incident server upon arrival (or left unmatched) and the algorithm is not allowed to reverse its decisions. Due to this no-reversal restriction, we are not able to guarantee an *exact* maximum matching in this model, only an approximate one.

Therefore, it is natural to study a different setting, where the top priority is to match as many clients as possible, and changes to the matching are possible but expensive. Formally, the goal is to always maintain a maximum matching while minimizing the number of changes made to the matching (denoted the *recourse*). This model is called the online model with recourse, and has been studied extensively over the past few years. For the specific problem of matching, the focus has been on vertex-arrival model, where clients arrive one at a time with all their edges. A recent result of Bernstein et al. [1] gives an upper bound of  $O(n \log^2 n)$  recourse for the case of general bipartite graphs. For trees the best known bound is  $O(n \log n)$  recourse, due to Bosek et al. [4]. These are nearly tight, as a lower bound of  $\Omega(n \log n)$  is known.

In this paper, we consider the more general model where all the vertices are known in advance, but the edges of the graph are revealed one at a time. Even for the simple case where the graph is a path, there is a lower bound of  $\Omega(n^2)$ . Therefore, we instead consider the natural relaxation where the graph is worst-case, but the edges are revealed in a *random* order. This relaxation is motivated by the fact that in many related models, such as the streaming setting or the standard online setting without recourse, faster algorithms have been obtained for the matching problem when the input comes in a random order. Our results are as follows:

- Our main result is that for the case of general (non-bipartite) graphs, the problem with random edge arrivals is almost as hard as in the adversarial setting: we show a family of graphs for which the expected recourse is  $\Omega\left(\frac{n^2}{\log n}\right)$ .
- We show that for some special cases of graphs, random arrival is significantly easier. For the case of trees, we get an upper bound of  $O(n \log^2 n)$  on the expected recourse. For the case of paths, this upper bound is  $O(n \log n)$ . We also show that the latter bound is tight, i.e. that the expected recourse is at least  $\Omega(n \log n)$ .

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms

**Keywords and phrases** matchings, edge-arrival, online model

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.11

**Funding** *Aaron Bernstein*: This work was done while funded by NSF Award 1942010 and the Simon's Group for Algorithms & Geometry.



© Aaron Bernstein and Aditi Dudeja;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 11; pp. 11:1–11:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

The online matching problem models a scenario in which a set of servers is given in advance, and a set of clients arrive one at a time, with each client incident to some of the servers. In the standard version of this model, the arriving client must be immediately matched to a free server or be left unmatched, and this decision is irrevocable. Due to this constraint, it is not possible to guarantee an *exact* matching, so the goal is to guarantee the best possible approximation. (See the work of Karp et al. [14], which shows that we can't get better than  $1 - \frac{1}{e}$  approximation.)

But there are several applications where the top priority is to match *all* the clients (or at least to have a maximum matching), and the irreversibility condition of the standard online model is too restrictive; in applications such as streaming content delivery, web hosting, job scheduling, or remote storage it is preferable to reallocate the clients provided the number of reallocations is small (see [5] for more details). Therefore, over the past decade there have been many papers on the so-called online model with *recourse*, where the goal is to maintain an *exact* solution the problem, while making as few changes to this solution as possible.

In the case of matching in particular, existing results focus on the vertex-arrival model, which is analogous to the similar model in online matching without recourse. In this model, clients arrive one at a time and ask to be matched to a server. The algorithm is allowed to change the matching over time and must always maintain a maximum matching: the goal is then to minimize the total number of changes made to the matching, denoted the *recourse*. Note that the trivial recourse bound is  $O(n^2)$  ( $n$  changes per client), but one can do significantly better. This model has been studied extensively (see for example, [9, 5, 2, 3, 10, 4, 1]), and the state of the art is an upper bound of  $O(n \log^2 n)$  on the total recourse [1] in bipartite graphs. For the special case of trees, the best known upper bound is  $O(n \log n)$  due to [4]. These upper bounds nearly match the lower bound of  $\Omega(n \log n)$  for trees due to [9].

In this paper, we consider a more general model where the graph can be non-bipartite and, more importantly, the *edges* in the graph are revealed one at a time; the algorithm must again maintain a maximum matching at all times. Unfortunately, we have very strong lower bounds when the order in which the edges arrive is adversarial; even for the simplest possible case of a path,  $\Omega(n^2)$  recourse is necessary. To overcome this lower bound, we consider a natural relaxation of this model where the adversary can still choose the graph, but edges arrive in a *random* order. One of the motivations behind this relaxation is that in several related models, such as the online model without recourse or the streaming model, we have been able to get faster algorithms when the input is assumed to arrive in a random order rather than an adversarial order. (See [13, 16] for online model without recourse, and [15, 12, 8, 7] for the streaming model).

Our results show that for the case of trees and paths, we can do significantly better in the random edge-arrival model: in particular, we show an upper bound of  $O(n \log n)$  on the expected recourse in the case of paths (which we show is tight), and a bound of  $O(n \log^2 n)$  in the case of trees. But our main result is that in general graphs, the random arrival setting is provably almost as hard as the adversarial setting. We state our main results formally:

► **Theorem 1.** *For any  $n > 2^{16}$ , there is a (non-bipartite) graph  $G_n$  (described in Section 3.1) with  $n$  vertices and  $\Theta(n \log n)$  edges, such that if edges of the graph arrive in a random order, then the total expected recourse taken by **any** algorithm that maintains a maximum matching in the graph is  $\Omega\left(\frac{n^2}{\log n}\right)$ .*

► **Theorem 2.** *Let  $T$  be a tree on  $n$  vertices and let the edges of  $T$  arrive one at a time in a random order. Then, the expected total recourse taken by an algorithm that maintains a maximum matching in  $T$  is at most  $O(n \log^2 n)$ .*

► **Theorem 3.** *Let  $P$  be a path on  $n$  vertices, and let the edges of  $P$  arrive in a random order. The expected total recourse taken by an algorithm that maintains a maximum matching in  $P$  is  $O(n \log n)$ . Moreover, this bound is tight: the expected recourse taken by **any** algorithm is  $\Omega(n \log n)$ .*

► **Remark 4.** For the lower bounds of Theorems 1 and 3, when we say that *any* algorithm has the given lower bound on expected recourse, this bound holds even if the algorithm knows the random permutation in advance. That is, the lower bound holds even if the algorithm is optimal for every possible ordering of the edges.

► **Remark 5.** For the upper bounds in Theorems 2 and 3, the algorithm we use simply changes the matching along an augmenting path whenever such a path becomes available due to the insertion of some edge. If there are multiple augmenting paths the algorithm can take, it chooses between them arbitrarily; the upper bound holds regardless of the choice of path.

We prove our main result, Theorem 1 in Section 3. For proofs of Theorem 2 and 3 we refer the reader to the full version of the paper. We leave as an intriguing open problem whether our lower bound in Theorem 1 also holds for *bipartite* graphs, or whether these graphs allow for expected  $o(n^{2-\epsilon})$  recourse when edges arrive in a random order. See Section 4 for more details.

## 2 Preliminaries

Let  $G$  be an unweighted graph. A matching in  $G$  is a set of vertex-disjoint edges. Given any matching  $M$  of  $G$ , we say that a vertex  $v$  is matched if it is incident to an edge in  $M$ , and free otherwise. Given any two matchings  $M$  and  $M'$ , we use  $M \oplus M'$  to denote the symmetric difference. We study the model of online matching with recourse under random edge arrivals. In this model, the adversary fixes any graph  $G = (V, E)$  with  $m$  edges and  $n$  vertices. The vertex set is given in advance, but the edges arrive one at a time; the arrival order  $e_1, \dots, e_m$  is a *random* permutation of  $E$ . The goal of the algorithm is to maintain a sequence of matchings  $M_1, \dots, M_m$ , such that  $M_i$  is a maximum matching in the graph  $(V, \{e_1, \dots, e_i\})$ . The total recourse of the algorithm is  $\sum_{i=1}^{m-1} |M_i \oplus M_{i+1}|$ , which is the total number of changes made to the matching throughout the entire sequence of insertions.

Intuitively, an algorithm that minimizes recourse should only change the matching when the maximum matching in the graph increases in size. We formalize this intuition in the remainder of this section.

► **Definition 6.** *Define a sequence  $M_{i_0}^*, M_{i_1}^*, \dots, M_{i_n}^*$  to be *only-augmenting* if  $M_{i_0}^* = \emptyset$ , each  $M_{i_j}^*$  is a maximum matching in  $G_{i_j}$ , and each symmetric difference  $M_{i_j}^* \oplus M_{i_{j+1}}^*$  consists of a single augmenting path; that is,  $M_{i_j}^* \oplus M_{i_{j+1}}^*$  consists of an odd-length path  $P$  in  $\{e_1, \dots, e_{i_{j+1}}\}$  such that every second edge of  $P$  is in  $M_{i_j}^*$ , but the first and last edges of  $P$  are not in  $M_{i_j}^*$ . We say that an algorithm is *only augmenting* if the sequence of distinct matchings produced by the algorithm is *only-augmenting*; in other words, in the sequence of matchings produced by an *only-augmenting* algorithm, for every  $1 \leq i \leq m-1$ , either  $M_i = M_{i+1}$ , or  $M_i \oplus M_{i+1}$  consists of a single augmenting path.*

## 11:4 Online Matching with Recourse: Random Edge Arrivals

► **Definition 7.** Let  $r(\sigma)$  is the best recourse achievable on permutation  $\sigma$  by an algorithm that knows  $\sigma$  in advance, and let  $r^*(\sigma)$  be the best recourse achievable by an only-augmenting algorithm that knows  $\sigma$  in advance. (Knowing  $\sigma$  in advance allows the respective algorithms to pick the best possible matching sequence for permutation  $\sigma$ .)

► **Observation 8.** Using the above notation, we note that  $\mathbb{E}_\sigma[r(\sigma)]$  is a lower bound on the expected recourse of any algorithm, while  $\mathbb{E}_\sigma[r^*(\sigma)]$  is a lower bound on the expected recourse of any only-augmenting algorithm. The lower bound applies even if the algorithm knows  $\sigma$  in advance.

The following Lemma allows us to assume throughout the paper that we are working with an only-augmenting algorithm. The proof of this lemma is relegated to the full version of the paper.

► **Lemma 9.** Given any permutation  $\sigma$ , we have  $r(\sigma) = r^*(\sigma)$ .

We now restate our main Theorem with the above lemma in mind.

► **Theorem 10.**  $\mathbb{E}_\sigma[r^*(\sigma)] = \Omega(n^2 / \log(n))$

► **Observation 11.** Observation 8, Lemma 9 and Theorem 10 immediately imply Theorem 1.

The lower bound proof of Section 3 is devoted entirely to proving Theorem 10

### 3 Lower Bound on Expected Recourse in General Graphs

This section will be devoted to proving Theorem 10, the main result of our paper. Recall from the preliminaries that we can assume that the algorithm is only-augmenting (See Definition 6) and that it knows the entire permutation  $\sigma$  in advance. In other words, to prove Theorem 1, it is sufficient to prove Theorem 10.

Our proof will proceed as follows. In Section 3.1 we define our candidate graph  $G_n$  (we will refer to it as  $G$  from now). The main step will be to show that between the times when half the edges of the graph have arrived and a three-quarters of the edges have arrived, the graph induced by non-isolated vertices contains a perfect matching or a near perfect matching throughout (see Definition 15 for a definition of near perfect matching). We will then use this fact to prove Theorem 10.

► **Remark 12.** Before we describe our graph, we describe how we will go about proving the lower bound. Suppose that our algorithm is given graph  $G = (V, E)$  as input, where  $|E| = m$ . In our model this graph is revealed to our algorithm one edge at a time, with the edges arriving in the order prescribed by a random permutation  $\sigma$ . Suppose we look at the graph at time  $t < m$ , then  $G_t$ , the graph at time  $t$  has the same distribution as the subgraph of  $G$  obtained by randomly sampling  $t$  out of  $m$  edges. We will show that between the times when  $t = 0.5 \cdot m$  and  $t = 0.75 \cdot m$ ,  $G_t$  will contain a perfect or a near-perfect matching. To prove this, we will show (in Section 3.2) that the distribution of  $G_t$  can be approximated by the following distribution: graph obtained by sampling each edge of  $G$  independently with probability  $\frac{t}{m}$ . Finally, we will prove our aforementioned claims about this new distribution (Section 3.3).

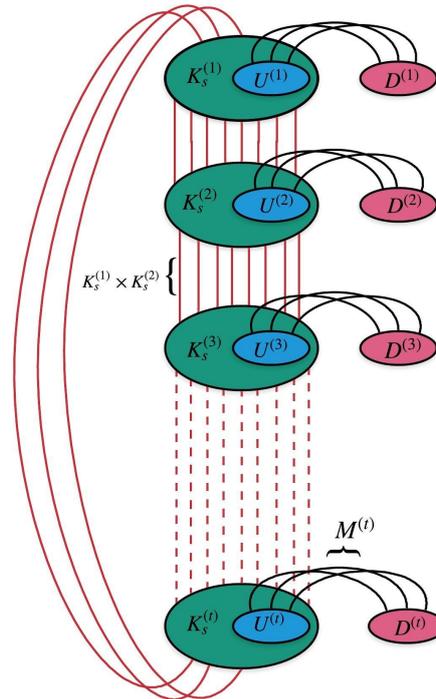
#### 3.1 The Graph

We use  $n$  to denote the number of vertices in our graph. In this write-up,  $s = 400 \log n$  and  $t = \frac{n}{500 \log n}$ . Let  $K_s$  denote the complete graph on  $s$  vertices. Our graph is called  $G$  (see Figure 1) and it consists of  $t$  copies of  $K_s$  that we index as  $K_s^{(i)}$  for  $1 \leq i \leq t$ . The

remaining  $\frac{n}{5}$  vertices are partitioned into  $t$  sets  $\{D^{(i)}\}_{1 \leq i \leq t}$  of size  $100 \log n$  each. The graph  $G$  contains the following edges.

1. For  $1 \leq i \leq t - 1$ , we introduce edges between every vertex of  $K_s^{(i)}$  and every vertex of  $K_s^{(i+1)}$ . Additionally, edges are also introduced between every vertex of  $K_s^{(1)}$  and every vertex of  $K_s^{(t)}$ .
2. For  $1 \leq i \leq t$ , we fix an arbitrary set  $U^{(i)} \subset K_s^{(i)}$  of size  $100 \log n$ . Introduce an arbitrary matching between  $U^{(i)}$  and  $D^{(i)}$ . Call this matching  $M^{(i)}$ . Let  $M = \cup_{i=1}^t M^{(i)}$ ; we add the edges of  $M$  to  $G$ . We also let  $U = \cup_{i=1}^t U^{(i)}$  and  $D = \cup_{i=1}^t D^{(i)}$ . For any  $u \in D \cup U$ , we define  $M(u)$  to be the vertex that  $u$  is matched to.

We denote the number of edges in  $G$  by  $m$ . Note that  $m = \Theta(n \log n)$ .



■ Figure 1 Graph  $G$ .

### 3.2 Relating $G_p$ and $G^{p \cdot m}$

► **Definition 13.** Let  $p \in [0, 1]$ . We define  $E_p \subset E(G)$  to be the set of edges obtained by sampling each  $e \in E(G)$  with independently probability  $p$ .

Let  $V_p = V(G) \setminus \{v \in D \text{ such that } (v, M(v)) \notin E_p\}$ ; note that  $V_p$  excludes isolated vertices in  $D$ . Let  $G_p$  be the graph with vertex set  $V_p$  and edge set  $E_p$ .

► **Definition 14.** Let  $E^{p \cdot m} \subset E(G)$  be the set of edges obtained by sampling  $p \cdot m$  random edges of  $E(G)$ . Let  $V^{p \cdot m} = V(G) \setminus \{v \in D \text{ such that } (v, M(v)) \notin E^{p \cdot m}\}$ ; note that  $V^{p \cdot m}$  excludes isolated vertices in  $D$ . Let  $G^{p \cdot m}$  be the graph with vertex set  $V^{p \cdot m}$  and the edge set  $E^{p \cdot m}$ .

► **Definition 15.** Let  $H$  be a graph with an odd number of vertices. Let  $\mathcal{M}$  be any matching of  $H$  that leaves exactly one vertex unmatched. Then,  $\mathcal{M}$  is called a near perfect matching of  $H$ .

## 11:6 Online Matching with Recourse: Random Edge Arrivals

We state the main theorem that we want to prove in this section:

► **Theorem 16.** *Let  $p \in \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ , then, the graph  $G^{p \cdot m}$  contains a perfect matching or a near perfect matching with probability at least  $1 - O\left(\frac{1}{n^3}\right)$ .*

To prove this theorem, we claim that it is sufficient to prove the following theorem:

► **Theorem 17.** *Let  $p \in \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ , then, graph  $G_p$  contains a matching or a near perfect matching with probability at least  $1 - O\left(\frac{1}{n^4}\right)$ .*

To show that Theorem 17 implies Theorem 16, we prove the following lemma:

► **Lemma 18.** *Let  $p \in \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ , and let  $G^{p \cdot m}$  and  $G_p$  be as described above, and let  $\mathcal{G}$  be the set of graphs that contain a perfect matching or a near perfect matching, then,*

$$\Pr(G^{p \cdot m} \notin \mathcal{G}) \leq 10\sqrt{m} \cdot \Pr(G_p \notin \mathcal{G}).$$

We refer the reader to the full version of this paper for a proof of Lemma 18. For now, we prove Theorem 16 assuming Theorem 17 and Lemma 18:

**Proof (Theorem 16).** It follows from Lemma 18 that:

$$\begin{aligned} \Pr(G^{p \cdot m} \text{ does not contain a matching}) &\leq 10\sqrt{m} \cdot \Pr(G_p \text{ does not contain a perfect matching}) \\ &= 10\sqrt{m} \cdot O\left(\frac{1}{n^4}\right) \quad (\text{Due to Theorem 17}) \\ &= O\left(\frac{1}{n^3}\right) \quad (\text{Since } m = \Theta(n \log n)). \quad \blacktriangleleft \end{aligned}$$

The following corollary follows from Theorem 16, via a union bound:

► **Corollary 19.** *Let  $I = \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ . Let  $\mathcal{G}$  be the sequence of graphs  $\{G^{p \cdot m}\}_{p \in I}$ . The probability that every  $G \in \mathcal{G}$  contains a perfect matching or a near perfect matching is at least  $1 - O\left(\frac{1}{n}\right)$ .*

The bulk of our paper is proving Theorem 17. But first, we provide some intuition for our choice of  $G$  by sketching how Corollary 19 implies our main result (Theorem 10).

**Proof sketch of Theorem 10.** Recall the edges  $M \subset E(G)$  which connect the vertices in  $D$ , where  $|M| = \Theta(n)$  (see 3.1). Consider how the graph  $G^{p \cdot m}$  evolves from for  $p = \frac{1}{2}$  to  $p = \frac{3}{4}$ . Let us assume without loss of generality that  $G^{\frac{1}{2} \cdot m}$  contains an even number of vertices. Whenever an edge  $(d, x)$  from  $M$  is inserted into the graph,  $d \in D$  is added to  $V(G^{p \cdot m})$  (See Definition 13). Since we know from Corollary 19 that  $G^{p \cdot m}$  contains a perfect matching whenever  $V(G^{p \cdot m})$  is even, we know that after every two edges  $(d, x)$  and  $(d', x')$  added to  $M$ , there is a perfect matching in the resulting graph; thus, the algorithm must take some augmenting path from  $d$  to  $d'$ . Because  $G$  consists of  $\Omega\left(\frac{n}{\log(n)}\right)$  consecutive layers, it is easy to see that with probability  $\frac{1}{2}$ , the shortest path from  $d$  to  $d'$  has length  $\Omega\left(\frac{n}{\log(n)}\right)$ . We expect to add  $\frac{|M|}{4} = \Omega(n)$  edges to  $M$  between  $G^{\frac{1}{2} \cdot m}$  and  $G^{\frac{3}{4} \cdot m}$ , so we have  $\Omega(n)$  augmenting paths of expected length  $\Omega\left(\frac{n}{\log(n)}\right)$ , which implies total augmenting path length  $\Omega\left(\frac{n^2}{\log(n)}\right)$ . See Section 3.5 for full proof.  $\blacktriangleleft$

### 3.3 Proving $G_p$ has a Near-Perfect Matching

We now turn to proving Theorem 17. To this end, we introduce some notation:

► **Definition 20.** Given  $G_p$ , we define the active subgraph  $A$  of  $G_p$  as follows: let  $V(A) = V(G_p) \setminus \{u \in D \cup U : (u, M(u)) \in G_p\}$ . The active subgraph  $A$  is the induced subgraph  $G_p[V(A)]$ .

► **Definition 21.** We define  $A^{(i)}$  to be the following subgraph of  $G_p$ : let  $V(A^{(i)}) = V(A) \cap V(K_s^{(i)})$  for  $1 \leq i \leq t$ . Let  $A^{(i)} = G_p[V(A^{(i)})]$  For  $1 \leq i \leq t$ , let  $|V(A^{(i)})| = a_i$ . Then,

1. If  $a_i$  is even, then let  $P^{(i)} \cup Q^{(i)}$  be an arbitrary  $\frac{a_i}{2}$  by  $\frac{a_i}{2}$  bipartition of  $V(A^{(i)})$ .
2. If  $a_i$  is odd, then let  $v^{(i)}$  be an arbitrary vertex in  $V(A^{(i)})$  and let  $P^{(i)} \cup Q^{(i)}$  be an arbitrary  $\lfloor \frac{a_i}{2} \rfloor$  by  $\lfloor \frac{a_i}{2} \rfloor$  bipartition of  $V(A^{(i)}) \setminus v^{(i)}$ .

We denote  $G(P^{(i)}, Q^{(i)})$  to be the bipartite graph between  $P^{(i)}$  and  $Q^{(i)}$ , with edge set  $E(P^{(i)}, Q^{(i)}) = (P^{(i)} \times Q^{(i)}) \cap E(A^{(i)})$

▷ **Claim 22.** We observe that  $V(A) \cap D = \emptyset$ . This follows from the following two facts:

1. Consider any  $u \in D$  such that  $(u, M(u)) \notin G_p$ . Then,  $u \notin V(G_p)$ . This follows immediately from Definition 13.
2. By Definition 20, we know that any  $u$  such that  $(u, M(u)) \in G_p$  is not included in  $V(A)$ .

▷ **Claim 23.** From Definition 20, we know that  $a_i \geq 400 \log n - |U^{(i)}|$ . Since  $|U^{(i)}| = 100 \log n$  (see Section 3.1 2), it follows that  $a_i \geq 300 \log n$ .

In order to prove Theorem 17, it is sufficient to prove the following theorem:

► **Theorem 24.** The active subgraph,  $A$  contains a perfect matching or a near perfect matching with probability at least  $1 - O\left(\frac{1}{n^4}\right)$ .

**Proof (Theorem 17).** Given a perfect (resp. near-perfect) matching  $\mathcal{M}(A)$  of  $A$ , we will construct a perfect (resp. near perfect) matching  $\mathcal{M}(G_p)$  of  $G_p$ . Consider any  $u \in V(G_p) \setminus V(A)$ . Note that  $M(u) \in V(G_p) \setminus V(A)$  and  $(u, M(u)) \in G_p$ . So we may match  $u$  to  $M(u)$  in  $G_p$ . In particular,  $\mathcal{M}(G_p) = \mathcal{M}(A) \cup \{(u, M(u)) \text{ where } u \in V(G_p) \setminus V(A)\}$ . Thus,  $\mathcal{M}(G_p)$  is a perfect (or a near perfect matching) of  $G_p$  if  $\mathcal{M}(A)$  is a perfect (or a near perfect matching) of  $A$ . ◀

### 3.4 Near Perfect Matching in Active Subgraph

To prove Theorem 24, we need Chernoff bound, and some existing results on matchings in random bipartite graphs.

► **Theorem 25.** [11] Define  $B(n, n, p)$  to be the bipartite graph obtained by deleting edges from  $K_{n,n}$  independently with probability  $1 - p$ . Then,

$$\Pr(B(n, n, p) \text{ does not contain a perfect matching}) = O(ne^{-np}).$$

► **Theorem 26 (Chernoff Bounds).** Let  $X_0, \dots, X_k$  be 0-1 random variables that are independent. Let  $\mu = \mathbb{E}\left[\sum_{i=1}^k X_i\right]$ . Then, for any  $0 \leq \delta \leq 1$ ,

$$\Pr\left(\sum_{i=1}^k X_i \leq (1 - \delta)\mu\right) \leq e^{-\frac{\delta^2 \mu}{2}} \text{ and,} \quad (1)$$

$$\Pr\left(\sum_{i=1}^k X_i \geq (1 + \delta)\mu\right) \leq e^{-\frac{\delta^2 \mu}{3}}. \quad (2)$$

Consider the  $A^{(i)}$ 's in Definition 21. We mentioned that for some of these  $A^{(i)}$ 's the corresponding  $a_i$ 's might be odd. Let  $\{A^{(i_1)}, \dots, A^{(i_k)}\}$  be this set, with  $i_1 < \dots < i_k$ . Let  $v^{(i_j)}$  be the vertex left out of the bipartition  $P^{(i_j)} \cup Q^{(i_j)}$  of  $A^{(i_j)}$  for  $1 \leq j \leq k$  (see Definition 21.2). We define the following events:

► **Definition 27.** For  $1 \leq i \leq t$ , let  $\mathcal{A}_i$  be the event that  $G(P^{(i)}, Q^{(i)})$  contains a perfect matching (or a near perfect matching). Let  $\mathcal{A} = \bigcap_{i=1}^t \mathcal{A}_i$ .

► **Definition 28.** Let  $\mathcal{M}'_i$  be a maximum matching of  $G(P^{(i)}, Q^{(i)})$  for  $1 \leq i \leq t$ . Let  $\mathcal{M}' = \bigcup_{i=1}^t \mathcal{M}'_i$ .

► **Definition 29.** For  $1 \leq m \leq \lfloor \frac{k}{2} \rfloor$ , let  $\mathcal{B}_m$  be the event that there is an augmenting path between  $v^{(i_{2m-1})}$  and  $v^{(i_{2m})}$  with respect to  $\mathcal{M}'$  in  $A$ . Let  $\mathcal{B} = \bigcap_{i=1}^{\lfloor \frac{k}{2} \rfloor} \mathcal{B}_m$ .

In order to prove Theorem 24, we follow these steps:

1. We will prove that each  $\mathcal{A}_i$  happens with high probability, and therefore by union bound,  $\mathcal{A}$  happens with high probability also.
2. We prove that each  $\mathcal{B}_m$ , conditioned on  $\mathcal{A}$  happens with high probability, and by union bound,  $\mathcal{B}$  conditioned on  $\mathcal{A}$  also happens with high probability.

In order to prove 2, we will show that for each  $1 \leq m \leq \lfloor \frac{k}{2} \rfloor$  there is an augmenting path between  $v^{(i_{2m-1})}$  and  $v^{(i_{2m})}$  which only consists of vertices between layers  $i_{2m-1}$  and  $i_{2m}$ . Therefore, these augmenting paths are vertex-disjoint from each other. These paths can be augmented simultaneously since they don't interfere with each other. So, 1 and 2 combined with this fact imply that the active graph,  $A$  contains a perfect matching or a near perfect matching with high probability.

Before we move on to proving 1 and 2, we note that  $G(P^{(i)}, Q^{(i)})$  and  $V(A^{(i)})$  are both random variables. In particular,  $V(A^{(i)}) = (V(K_S^{(i)}) \setminus U^{(i)}) \cup S$ , where  $S$  is a random subset of  $U^{(i)}$  obtained by excluding every vertex with probability  $p$ . However, if we fix the vertex set  $V(A^{(i)})$ , then the edges of  $G(P^{(i)}, Q^{(i)})$  have the same distribution as that of a random bipartite graph; we remind the reader that  $P^{(i)} \cup Q^{(i)}$  is an arbitrary bipartition of  $A^{(i)}$  (see Definition 21). Formally:

► **Observation 30.** For  $1 \leq i \leq t$ ,  $G(P^{(i)}, Q^{(i)})$  conditioned on  $V(A^{(i)}) = S$ , where  $|S| = a_i$ , has the same distribution as  $B(\lfloor \frac{a_i}{2} \rfloor, \lfloor \frac{a_i}{2} \rfloor, p)$ .

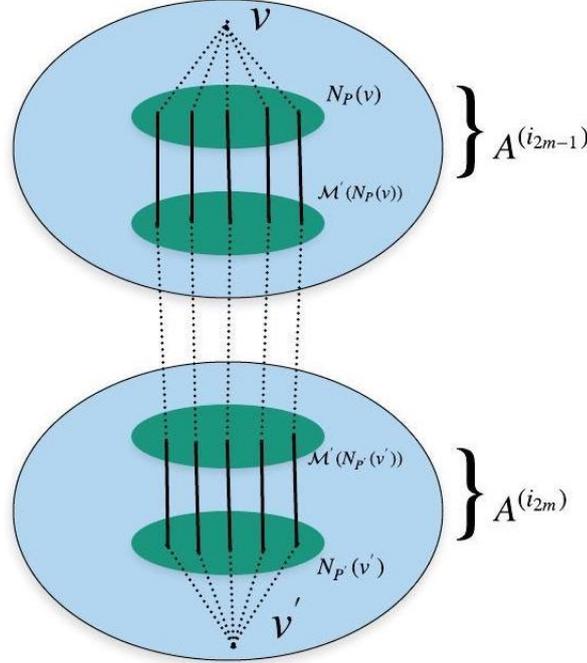
Now we prove the following lemma:

► **Lemma 31.** For  $1 \leq i \leq t$ ,  $\Pr(\neg \mathcal{A}_i) = O(\frac{1}{n^5})$ . Moreover,  $\Pr(\neg \mathcal{A}) = O(\frac{1}{n^4})$ .

**Proof.** We know that:

$$\begin{aligned}
 \Pr(\neg \mathcal{A}_i) &= \sum_T \Pr(\neg \mathcal{A}_i \mid V(A^{(i)}) = T) \cdot \Pr(V(A^{(i)}) = T) \\
 &= \sum_T O(|T| \cdot e^{-|T| \cdot p}) \cdot \Pr(V(A^{(i)}) = T) \text{ (Due to Observation 30 and Lemma 25)} \\
 &= \sum_T O\left(\frac{1}{n^5}\right) \cdot \Pr(V(A^{(i)}) = T) \text{ (Due to Claim 23 that } a_i \geq 300 \log n \text{ and } p \geq 0.5) \\
 &= O\left(\frac{1}{n^5}\right) \text{ (Since we are summing over disjoint events).}
 \end{aligned}$$

By union bound it follows that,  $\Pr(\neg \mathcal{A}) = O(\frac{1}{n^4})$ . ◀



■ **Figure 2** Case 1: When unmatched vertices are in consecutive layers.

► **Theorem 32.** For  $1 \leq m \leq \lfloor \frac{k}{2} \rfloor$ ,  $\Pr(\neg \mathcal{B}_m \mid \mathcal{A}) = O\left(\frac{1}{n^8}\right)$ . Therefore, by union bound it follows that  $\Pr(\neg \mathcal{B} \mid \mathcal{A}) = O\left(\frac{1}{n^7}\right)$ .

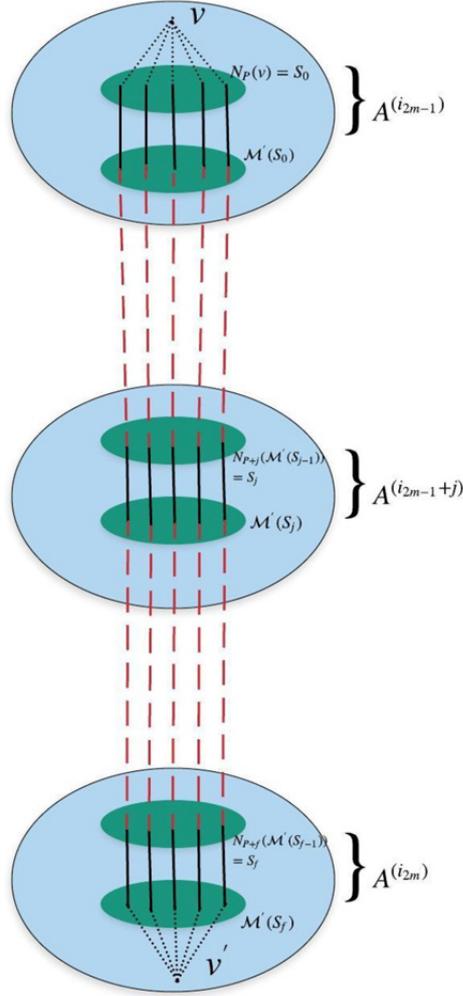
**Proof.** To bound  $\Pr(\neg \mathcal{B}_m \mid \mathcal{A})$ , we consider two cases:

1. **Case 1:  $v_{i_{2m-1}}$  and  $v_{i_{2m}}$  are in consecutive layers.** That is,  $i_{2m} = i_{2m-1} + 1$ . We will give an overview of what we are about to do. We will use  $v$  to denote  $v_{i_{2m-1}}$ ,  $v'$  to denote  $v_{i_{2m}}$ ,  $P$  and  $P'$  to denote  $P^{(i_{2m-1})}$  and  $P^{(i_{2m})}$ ,  $Q$  and  $Q'$  to denote  $Q^{(i_{2m-1})}$  and  $Q^{(i_{2m})}$  respectively.

► **Definition 33.** Let  $N_P(v)$  (resp.  $N_{P'}(v')$ ) denote the set of vertices in  $P$  (resp.  $P'$ ) adjacent to  $v$  (resp.  $v'$ ). Let  $\deg_P(v)$  (resp.  $\deg_{P'}(v')$ ) denote  $|N_P(v)|$  (resp.  $|N_{P'}(v')|$ ).

For a set of vertices  $S$ , let  $\mathcal{M}'(S)$  denote the set of vertices matched to  $S$  in  $\mathcal{M}'$  (refer to Definition 28 for the definition of  $\mathcal{M}'$ ). We will prove that with high probability  $|\mathcal{M}'(N_P(v))|$  and  $|\mathcal{M}'(N_{P'}(v'))|$  are large. Conditioned on these sizes being large, we will prove that there is an edge  $(x, x')$  in  $A$  where  $x \in \mathcal{M}'(N_P(v))$  and  $x' \in \mathcal{M}'(N_{P'}(v'))$ . It follows there is an augmenting path  $\mathcal{P} = (v, \mathcal{M}'(x), x, x', \mathcal{M}'(x'), v')$  in  $A$  (note that  $\mathcal{M}'(x) \in N_P(v)$  and  $\mathcal{M}'(x') \in N_{P'}(v')$ ). (See Figure 2)

To show this, we first show that  $|N_P(v)|$  and  $|N_{P'}(v')|$  are large with high probability. We will condition on  $\mathcal{A}$ , so  $|\mathcal{M}'(N_P(v))|$  and  $|\mathcal{M}'(N_{P'}(v'))|$  will consequently be large with high probability. It then follows that one of the edges between these two sets is in  $A$  with high probability.



■ **Figure 3** Case 2: When  $v$  and  $v'$  are not in consecutive layers.

We now turn to the formal proof of case 1. Let  $X_v$  and  $X_{v'}$  be the random variables denoting  $\deg_P(v)$  and  $\deg_{P'}(v')$  respectively (see Definition 33). Each edge incident on  $v$  and  $v'$  in  $\mathcal{A}$  is sampled independently with probability  $p \in [0.5, 0.75]$ . This is true even if we condition on the event  $\mathcal{A}$ . Consequently,  $\mathbb{E}[X_v | \mathcal{A}] = \mathbb{E}[X_{v'} | \mathcal{A}] \geq 75 \log n$ . Since  $X_v$  is the sum of 0 – 1 independent random variables, we may apply Chernoff bound (see Theorem 26). It follows that:

$$\Pr(X_v \leq 25 \log n | \mathcal{A}) = O\left(\frac{1}{n^8}\right).$$

Similarly, we have:

$$\Pr(X_{v'} \leq 25 \log n | \mathcal{A}) = O\left(\frac{1}{n^8}\right).$$

Define  $\mathcal{Y}$  to be the event that  $|\mathcal{M}'(N_P(v))| \geq 25 \log n$  and  $|\mathcal{M}'(N_{P'}(v'))| \geq 25 \log n$ .

Observe that,

$$\Pr(\neg\mathcal{Y} \mid \mathcal{A}) \leq \Pr(X_v \leq 25 \log n \mid \mathcal{A}) + \Pr(X_{v'} \leq 25 \log n \mid \mathcal{A}) = O\left(\frac{1}{n^8}\right).$$

Define  $\mathcal{Z}$  to be the event that there is an edge between  $\mathcal{M}'(N_P(v))$  and  $\mathcal{M}'(N_{P'}(v'))$ . Observe that,

$$\Pr(\neg\mathcal{Z} \mid \mathcal{A}) \leq \Pr(\neg\mathcal{Y} \mid \mathcal{A}) + \Pr(\neg\mathcal{Z} \mid \mathcal{Y}, \mathcal{A}) = O\left(\frac{1}{n^8}\right) + \frac{1}{n^{O(\log n)}}.$$

The second term follows from the fact that each edge appears independently with probability  $p \in [0.5, 0.75]$ , and there are  $\Omega(\log^2 n)$  edges between  $\mathcal{M}'(N_P(v))$  and  $\mathcal{M}'(N_{P'}(v'))$  conditioned on  $\mathcal{Y}$ . It follows that  $\Pr(\neg\mathcal{B}_m \mid \mathcal{A}) \leq \Pr(\neg\mathcal{Z} \mid \mathcal{A}) = O\left(\frac{1}{n^8}\right)$ . This proves our claim for this case.

2. **Case 2:  $i_{2m} > i_{2m-1} + 1$ .** We denote  $v_{i_{2m-1}}$  by  $v$ ,  $P^{(i_{2m-1})}$  by  $P$  and  $v^{(i_{2m})}$  by  $v'$ . Let  $f = i_{2m} - i_{2m-1}$ . For  $1 \leq j \leq f$ , let  $P^{(i_{2m-1}+j)}$  be denoted by  $P+j$ . We similarly define  $Q$  and  $Q+j$  (see Figure 3). We also define the following sets:

$$\begin{aligned} S_0 &= N_P(v) \\ S_j &= N_{P+j}(\mathcal{M}'(S_{j-1})) \text{ for } 1 \leq j \leq f. \end{aligned}$$

For  $0 \leq j \leq f$ , let  $\mathcal{X}_j$  be the event that  $|\mathcal{M}'(S_j)| \geq 25 \log n$ . Let  $\mathcal{E}$  be the event that there is an edge between  $v'$  and  $\mathcal{M}'(S_f)$ . It is easy to check that the occurrence of  $\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_f$  implies that there is an alternating path from  $v$  to a large set of vertices (at least  $\Omega(\log n)$ ) in  $Q+j$  for all  $j \in [f]$ . Note that  $\mathcal{E}$  implies that there is an edge from  $Q+f$  to  $v'$ . Combined,  $\mathcal{X}_1, \dots, \mathcal{X}_f, \mathcal{E}$  imply an augmenting path from  $v$  to  $v'$ . We thus have:

► **Observation 34.** Let  $\mathcal{B}_m$  and  $\mathcal{X}_1, \dots, \mathcal{X}_f, \mathcal{E}$  be as defined above (refer to Definition 29 for a definition of  $\mathcal{B}_m$ ), then:

$$\Pr(\mathcal{B}_m \mid \mathcal{A}) \geq \Pr\left(\bigcap_{k=0}^f \mathcal{X}_k \cap \mathcal{E} \mid \mathcal{A}\right).$$

From the above observation, we deduce that in order to upper bound  $\Pr(\neg\mathcal{B}_m \mid \mathcal{A})$ , it is sufficient to upper bound  $\Pr\left(\bigcup_{k=0}^f \neg\mathcal{X}_k \cup \neg\mathcal{E} \mid \mathcal{A}\right)$ . We know that:

$$\Pr\left(\bigcup_{k=0}^f \neg\mathcal{X}_k \cup \neg\mathcal{E} \mid \mathcal{A}\right) \leq \sum_{k=0}^f \Pr(\neg\mathcal{X}_k \mid \bigcap_{k=0}^{i-1} \mathcal{X}_k \cap \mathcal{A}) + \Pr(\neg\mathcal{E} \mid \bigcap_{k=0}^f \mathcal{X}_k \cap \mathcal{A}).$$

(Follows from the definition of conditional probability)

We computed  $\Pr(\neg\mathcal{X}_0 \mid \mathcal{A})$  in case 1. We remind the reader this is just the probability that  $|\mathcal{M}'(S_0)| \leq 25 \log n$ . We now show how to compute  $\Pr(\neg\mathcal{X}_j \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1})$ . Consider any  $w \in P+j$ . We want to compute the probability that  $w$  is in the set  $N_{P+j}(\mathcal{M}'(S_{j-1})) = S_j$  conditioned on the events  $\mathcal{X}_{j-1}$  and  $\mathcal{A}$ . Since every edge on  $w$  is present in the active graph  $A$  independently with probability  $p$ :

$$\Pr(w \notin S_j \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1}) \leq (1-p)^{25 \log n} \quad (3)$$

$$\leq \left(\frac{1}{2}\right)^{25 \log n} \quad (\text{Due to the fact that } p \geq 0.5). \quad (4)$$

## 11:12 Online Matching with Recourse: Random Edge Arrivals

This implies that:

$$\mathbb{E}[|S_j| \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1}] \geq 100 \log n.$$

Since  $|S_j|$  is a sum of  $0-1$  random variables (it is the sum of  $\mathbb{1}_{\{v \in S_j\}}$ , that take value 0 with probability  $O\left(\frac{1}{n^{25}}\right)$  (due to Equation (4)) and 1 otherwise), we can apply Chernoff bounds (Theorem 26):

$$\Pr(|S_j| \leq 25 \log n \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1}) = O\left(\frac{1}{n^9}\right).$$

Since we condition on  $\mathcal{A}$  (that is a perfect or, a near perfect matching being present), we know that:

$$|\mathcal{M}'(S_j)| = |S_j|$$

Consequently, we have:

$$\begin{aligned} \Pr(|\mathcal{M}'(S_j)| \leq 25 \log n \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1}) &= \Pr(|S_j| \leq 25 \log n \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_{j-1}) \\ &= O\left(\frac{1}{n^9}\right). \end{aligned}$$

Finally, we want to bound  $\Pr(\neg \mathcal{E} \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_f)$ . This can be upper bounded:

$$\begin{aligned} \Pr(\neg \mathcal{E} \mid \mathcal{A}, \mathcal{X}_0, \dots, \mathcal{X}_f) &\leq \left(\frac{1}{2}\right)^{25 \log n} \\ &\quad \text{(Edges on } v' \text{ appear independently with probability } p \geq 0.5) \\ &= O\left(\frac{1}{n^{25}}\right). \end{aligned}$$

It is immediate from Observation 34 that:

$$\Pr(\neg \mathcal{B}_m \mid \mathcal{A}) = O\left(\frac{1}{n^8}\right).$$

From case 1 and case 2, we know that by union bound,  $\Pr(\neg \mathcal{B} \mid \mathcal{A}) = O\left(\frac{1}{n^7}\right)$ . ◀

**Proof (Theorem 24).** From Lemma 31 and Theorem 32 we have that:

$$\Pr(A \text{ does not contain a perfect matching}) \leq \Pr(\neg \mathcal{A}) + \Pr(\neg \mathcal{B} \mid \mathcal{A}) = O\left(\frac{1}{n^4}\right). \quad \blacktriangleleft$$

### 3.5 Lower Bound On Lengths of Augmenting Paths

We start with some definitions:

► **Definition 35.** For  $i \in \{1, \dots, m\}$ , we denote by  $e_i$ , the edges arriving at time  $i$ . Let  $S = \{e_{0.5m}, \dots, e_{0.75m}\}$ .

This section will be devoted to proving that among the edges in  $S$ ,  $\Omega(n)$  edges will join augmenting paths of expected length  $\Omega\left(\frac{n}{\log n}\right)$ , and the algorithm is forced to augment along these. Formally,

► **Theorem 36.** With high probability, there exists  $S' \subset S$ ,  $|S'| \geq \frac{n}{100}$  such that each  $e \in S'$  joins an augmenting path of expected length at least  $\Omega\left(\frac{n}{\log n}\right)$ .

We first give a proof of Theorem 10 using Theorem 36:

**Proof (Theorem 10).** For  $i \in [m]$ , let  $\mathcal{Z}_i$  be the random variable denoting the length of the augmenting path that we augment along when the edge  $e_i$  joins. Let  $\mathcal{Z} = \sum_{i=1}^m \mathcal{Z}_i$ , which is the random variable denoting the total length of the augmenting paths taken during the course of the algorithm. We want to compute the quantity  $\mathbb{E}[\mathcal{Z}]$ . We note that:

$$\begin{aligned} \mathbb{E}[\mathcal{Z}] &= \sum_{i=1}^m \mathbb{E}[\mathcal{Z}_i] \geq \sum_{j \in S'} \mathbb{E}[\mathcal{Z}_j] = |S'| \cdot \Omega\left(\frac{n}{\log n}\right) \\ &= \Omega\left(\frac{n^2}{\log n}\right). \end{aligned}$$

(Due to Theorem 36) ◀

Before we prove Theorem 36, we need certain observations, and the following version of Chernoff for negatively associated random variables:

► **Theorem 37.** [6] Let  $X_0, \dots, X_k$  be 0–1 random variables that are negatively associated. Let  $\mu = \mathbb{E}\left[\sum_{i=1}^k X_i\right]$ . Then, for any  $0 \leq \delta \leq 1$ ,

$$\Pr\left(\sum_{i=1}^k X_i \leq (1 - \delta)\mu\right) \leq e^{-\frac{\delta^2 \mu}{2}} \text{ and,} \quad (5)$$

$$\Pr\left(\sum_{i=1}^k X_i \geq (1 + \delta)\mu\right) \leq e^{-\frac{\delta^2 \mu}{3}}. \quad (6)$$

We remind the reader of the edges  $M$  in graph  $G$  between  $D$  and  $U$  (refer to Section 3.12). Note that  $|M| \geq \frac{n}{5}$ . Further,  $M = \cup_{i=1}^t M^{(i)}$ , and  $|M^{(i)}| \geq 100 \log n$  for all  $i \in [t]$ .

We now prove the following claim about  $S$ :

▷ **Claim 38.** Let  $\mathcal{R}$  be the event that for all  $i \in [t]$ ,  $|M^{(i)} \cap S| \geq 10 \log n$ . Then,  $\Pr(\mathcal{R}) \geq 1 - O\left(\frac{1}{n^3}\right)$ .

Proof. Consider any  $M^{(i)}$ , and let  $e \in M^{(i)}$ . Let  $Z_e$  be a 0–1 random variable that takes value 1 if  $e \in S$ , and 0 otherwise. Let  $Z = \sum_{e \in M^{(i)}} Z_e$ . This is the random variable that denotes  $|M^{(i)} \cap S|$ . Further,  $Z$  is a sum of negatively associated random variables, and therefore obeys the condition of Theorem 37. We note the following:

$$\Pr(Z_e = 1) = \frac{1}{4} \text{ and, } \mathbb{E}[Z] = 25 \log n.$$

It follows that:

$$\Pr(Z \leq 10 \log n) \leq \exp\left(- (0.6)^2 (0.5) 25 \log n\right) \leq \exp(-4.5 \log n) = O\left(\frac{1}{n^4}\right).$$

Due to union bound, we know that  $\Pr(\mathcal{R}) \geq 1 - O\left(\frac{1}{n^3}\right)$ . ◀

We also have the following corollary due to Claim 38:

► **Corollary 39.** With probability at least  $1 - O\left(\frac{1}{n^3}\right)$ ,  $|M \cap S| \geq \frac{n}{50}$ .

We are ready to define the candidate set  $S'$  in Theorem 36, but before that we give a final definition:

## 11:14 Online Matching with Recourse: Random Edge Arrivals

► **Definition 40.** Consider edges  $e \in M^{(i)}$  and  $f \in M^{(j)}$  (see 3.12 for the definition of  $M^{(i)}$ ). Then let  $d(e, f) = \min \{t - |i - j|, |i - j|\}$ .

Let  $M \cap S = \{e_{i_1}, \dots, e_{i_q}\}$ . Let us assume without loss of generality that before the arrival of  $e_{i_1}$ , the set  $V(G_{i_1-1})$  is even, so by Theorem 16 the graph  $G_{i_1-1}$  has a perfect matching. We define  $S'$  to contain every second edge of  $M \cap S$ : that is,  $S' = \left\{ e_{i_2}, e_{i_4}, \dots, e_{i_{2\lfloor \frac{q}{2} \rfloor}} \right\}$ . For the rest of the proof we proceed as follows: we will show that with high probability, when  $e_{i_{2s}}$  arrives, it will join an augmenting path ending at  $e_{i_{2s-1}}$  where  $s \in \{1, \dots, \lfloor \frac{q}{2} \rfloor\}$ . Let  $e_{i_{2s}} \in M^{(j)}$  and  $e_{i_{2s+1}} \in M^{(j')}$ . Then, the length of the augmenting path that  $e_{i_{2s-1}}$  joins is at least  $d(e_{i_{2s-1}}, e_{i_{2s}}) = \min \{t - |j' - j|, |j' - j|\}$ . We prove that the expected value of this quantity is at least  $\Omega\left(\frac{n}{\log n}\right)$ .

We prove the following observation:

► **Lemma 41.** Let  $e$  and  $f$  be two edges that are chosen uniformly at random from  $M$ . Then,  $\mathbb{E}[d(e, f)] \geq \frac{n}{2000 \log n}$ .

**Proof.** The total number of possible choices for  $(e, f) = \left(\frac{n}{5}\right) \cdot \left(\frac{n}{5} - 1\right)$ . The total number choices for  $(e, f)$  such that  $d(e, f) = k$ , are  $\left(\frac{n}{500 \log n}\right) \cdot (100 \log n) \cdot (200 \log n)$ . To see this, fix a layer for  $e$ , then the number of choices of  $f$  for which  $d(e, f) = k$  are exactly  $200 \log n$ . Finally, the total number possible choices of layers for  $e$  is  $\frac{n}{500 \log n}$ . This implies that:

$$\begin{aligned} \Pr(d(e, f) = k) &= \frac{\left(\frac{n}{500 \log n}\right) \cdot (100 \log n) \cdot (200 \log n)}{\left(\frac{n}{5}\right) \cdot \left(\frac{n}{5} - 1\right)} \\ &\geq \frac{\left(\frac{n}{500 \log n}\right) \cdot (100 \log n) \cdot (200 \log n)}{\left(\frac{n}{5}\right) \cdot \left(\frac{n}{5}\right)} \\ &\geq \frac{1000 \log n}{n}. \end{aligned}$$

Finally, we have that:

$$\mathbb{E}[d(e, f)] = \sum_{k=0}^{\frac{t}{2}} k \cdot \Pr(d(e, f) = k) \geq \frac{t}{4} = \frac{n}{2000 \log n}. \quad \blacktriangleleft$$

We state an immediate corollary of Lemma 41:

► **Corollary 42.** For all  $s \in \{1, \dots, \lfloor \frac{q}{2} \rfloor\}$ ,  $\mathbb{E}[d(e_{i_{2s-1}}, e_{i_{2s}})] \geq \frac{n}{2000 \log n}$ .

► **Lemma 43.** If  $G^{p \cdot m}$  contains a perfect matching or a near perfect matching for all  $p \in \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ , then for all  $s \in \{1, \dots, \lfloor \frac{q}{2} \rfloor\}$ ,  $e_{i_{2s}}$  joins an augmenting path that ends in  $e_{i_{2s-1}}$ .

**Proof.** We remind the reader that  $|V(G^{p \cdot m})|$  is a random variable (check Definition 14) and it's value increases if and only if the edges in  $M$  arrive. Recall the assumption that  $|V(G_{i_1-1})|$  is even. Upon the arrival of  $e_{i_1}$ , we have a near perfect matching in the graph, and this remains the case until  $e_{i_2}$  arrives. At this point under our assumption, there must be a perfect matching in the graph. However, the matching that is currently maintained by the algorithm leaves the vertices are the end points of  $e_{i_1}$  and  $e_{i_2}$  in  $D$  unmatched. (Here we use the simplifying assumption from the preliminaries that the algorithm is only-augmenting, so since the arrival of  $e_{i_1}$  does not increase the size of the maximum matching,

and since the algorithm only changes the matching via augmenting paths, the endpoint of  $e_{i_1}$  in  $D$  remains free until the arrival of  $e_{i_2}$ .) It follows that these endpoints are joined together by an augmenting path. Continuing this way, we can prove the theorem for any  $s \in \{1, \dots, \lfloor \frac{q}{2} \rfloor\}$ . ◀

**Proof (Theorem 36).** Let  $\mathcal{F}$  be the event that there is an  $S' \subset S$ ,  $|S'| \geq \frac{n}{100}$  such that each  $e \in S'$  augments along a path of expected length at least  $\Omega\left(\frac{n}{\log n}\right)$ . Note that the event  $\mathcal{F}$  fails to happen if one of these go wrong:

1.  $|S'| \leq \frac{n}{100}$ . We call this event  $\neg\mathcal{U}$ . We know from Corollary 39 that  $\Pr(\neg\mathcal{U}) = O\left(\frac{1}{n^3}\right)$ . This is because  $S'$  just takes alternate elements from  $S$ .
2. Let  $\mathcal{V}$  be the event that for all  $p \in \{0.5, \frac{0.5 \cdot m + 1}{m}, \dots, \frac{0.75 \cdot m - 1}{m}, 0.75\}$ ,  $G^{p \cdot m}$  contain a perfect matching or a near perfect matching. Then, from Lemma 43 we know that  $\mathcal{V}$  implies that for all  $s \in \{1, \dots, \lfloor \frac{q}{2} \rfloor\}$ ,  $e_{i_{2s-1}}$  joins an augmenting path ending in  $e_{i_{2s}}$ . From Corollary 42, we know all these paths have expected length at least  $\frac{n}{2000 \log n}$ . We know from Corollary 19, that  $\Pr(\neg\mathcal{V}) = O\left(\frac{1}{n}\right)$ .

It follows that the occurrence of  $\mathcal{A}$  and  $\mathcal{B}$  implies the occurrence of  $\mathcal{F}$ . Consequently,  $\Pr(\mathcal{F}) \geq 1 - \Pr(\neg\mathcal{U}) - \Pr(\neg\mathcal{V}) \geq 1 - O\left(\frac{1}{n}\right)$ . ◀

## 4 Conclusion and Open Problems

We consider the problem of maximum matching with recourse in the random edge-arrival setting. The goal is to compute the expected recourse. As mentioned in the introduction, there are strong lower bounds of  $\Omega(n^2)$  in the adversarial edge-arrival model, even for the case of simple paths. For random edge-arrivals, we can do significantly better for special classes of graphs: we prove an upper bound of  $O(n \log n)$  for the case of paths and  $O(n \log^2 n)$  for the case of trees. This bound is tight up to  $\log n$  factors, since we prove that for the case of paths, any algorithm must take expected total recourse of  $\Omega(n \log n)$ . But for general graphs, we show that random arrival is basically as hard as adversarial arrival: we give a family of graphs for which the expected recourse is at least  $\Omega\left(\frac{n^2}{\log n}\right)$ .

An interesting open question is the case of *bipartite* graphs: if edge-arrivals are random, can we prove a similar lower bound of  $\Omega\left(\frac{n^2}{\text{polylog}(n)}\right)$  on the expected recourse? Our current lower-bound construction seems hard to extend to the bipartite case, as our proof crucially relies on the fact that after a constant fraction of the edges have arrived, if we focus only on the non-isolated vertices in the lower-bound graph  $G$ , then  $G$  contains a perfect matching with high probability. This allowed us to force the adversary to take an augmenting path between every new pair of non-isolated vertices. But in the case of bipartite graphs, it seems difficult to guarantee a perfect matching between the non-isolated vertices because the number of non-isolated vertices on the left might not be equal to the number on the right; in fact, they are likely to differ by a  $\Theta(\sqrt{n})$  factor.

---

## References

- 1 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized  $o(\log^2 n)$  replacements. *Journal of the ACM (JACM)*, 66(5):1–23, 2019.
- 2 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 384–393. IEEE, 2014.

- 3 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Shortest augmenting paths for online matchings on trees. In *International Workshop on Approximation and Online Algorithms*, pages 59–71. Springer, 2015.
- 4 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In *Latin American Symposium on Theoretical Informatics*, pages 201–216. Springer, 2018.
- 5 Kamalika Chaudhuri, Constantinos Daskalakis, Robert D Kleinberg, and Henry Lin. Online bipartite perfect matching with augmentations. In *IEEE INFOCOM 2009*, pages 1044–1052. IEEE, 2009.
- 6 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
- 7 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785. SIAM, 2020. doi:10.1137/1.9781611975994.108.
- 8 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 491–500, 2019.
- 9 Edward F Grove, Ming-Yang Kao, P Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In *Workshop on Algorithms and Data Structures*, pages 194–205. Springer, 1995.
- 10 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 468–479. SIAM, 2014.
- 11 Svante Janson, Tomasz Luczak, and Andrzej Rucinski. *Random graphs*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2000. doi:10.1002/9781118032718.
- 12 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 734–751. SIAM, 2014.
- 13 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.
- 14 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.
- 15 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 231–242. Springer, 2012.
- 16 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

# Hard QBFs for Merge Resolution

Olaf Beyersdorff 

Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany  
olaf.beyersdorff@uni-jena.de

Joshua Blinkhorn 

Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany  
joshua.blinkhorn@uni-jena.de

Meena Mahajan 

The Institute of Mathematical Sciences, HBNI, Chennai, India  
meena@imsc.res.in

Tomáš Peitl 

Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany  
tomas.peitl@uni-jena.de

Gaurav Sood 

The Institute of Mathematical Sciences, HBNI, Chennai, India  
gauravs@imsc.res.in

---

## Abstract

We prove the first proof size lower bounds for the proof system Merge Resolution (MRes [6]), a refutational proof system for prenex quantified Boolean formulas (QBF) with a CNF matrix. Unlike most QBF resolution systems in the literature, proofs in MRes consist of resolution steps *together* with information on countermodels, which are syntactically stored in the proofs as merge maps. As demonstrated in [6], this makes MRes quite powerful: it has strategy extraction by design and allows short proofs for formulas which are hard for classical QBF resolution systems.

Here we show the first *exponential lower bounds for MRes*, thereby uncovering limitations of MRes. Technically, the results are either transferred from bounds from circuit complexity (for restricted versions of MRes) or directly obtained by combinatorial arguments (for full MRes). Our results imply that the MRes approach is *largely orthogonal to other QBF resolution models* such as the QCDCL resolution systems QRes and QURes and the expansion systems  $\forall\text{Exp} + \text{Res}$  and IR.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Proof complexity

**Keywords and phrases** QBF, resolution, proof complexity, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.12

**Funding** *Olaf Beyersdorff*: John Templeton Foundation (grant no. 60842), Carl Zeiss Foundation.  
*Tomáš Peitl*: Grant J-4361 of the Austrian Science Fund FWF.

**Acknowledgements** Part of this work was done during the Dagstuhl Seminar “SAT and Interactions” (Seminar 20061).

## 1 Introduction

*Proof complexity* aims to provide a theoretical understanding of the ease or difficulty of proving statements formally. It also aims to explain the success stories of, as well as the obstacles faced by, algorithmic approaches to hard problems such as satisfiability (SAT) and Quantified Boolean Formulas (QBF) [18, 28]. While propositional proof complexity, the study of proofs of unsatisfiability of propositional formulas, has been around for decades [19, 26], the area of *QBF proof complexity* is relatively new, with theoretical studies gaining traction only in the last decade or so [2, 7, 9, 10]. While inheriting and using a wealth of techniques from propositional proof complexity [11, 13, 24], QBF proof complexity has also given several



© Olaf Beyersdorff, Joshua Blinkhorn, Meena Mahajan, Tomáš Peitl, and Gaurav Sood;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 12; pp. 12:1–12:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

new perspectives specific to QBF [5, 23, 34], and these perspectives and their connections to QBF solving [31, 38] as well as their practical applications [33] have driven the search for newer proof systems [1, 10, 21, 27, 29].

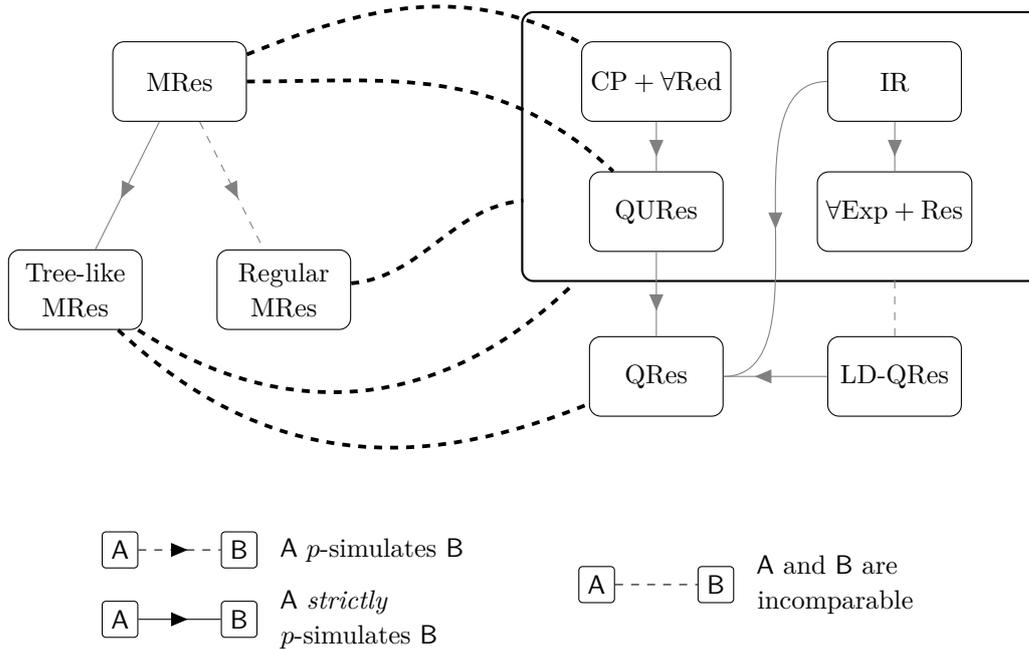
Many of the currently known QBF proof systems are built on the best-studied propositional proof system *resolution* [16, 32]. Broadly speaking, resolution has been adapted to handle the universal variables in QBFs in two intrinsically different ways. The first is an *expansion-based approach*: universal variables are eliminated at the outset by implicitly expanding the universal quantifiers into conjunctions, creating annotated copies of existential variables. The systems  $\forall\text{Exp} + \text{Res}$ , IR, and IRM [10, 23] are of this type. The second is a *reduction-rule approach*: under certain conditions, resolution may be blocked, and also under certain conditions, universal variables can be deleted from clauses. The conditions are formulated to preserve soundness, ensuring that if a QBF is true, then so is the QBF resulting from adding a derived clause. The systems QRes, QURes, CP +  $\forall\text{Red}$  [12, 25, 36] are of this type.

A central role in QBF proof complexity is played by the *two-player evaluation game* on QBFs, and the existence of winning strategies for the universal player in false QBFs. For many QBF resolution systems, such strategies were used to construct proofs and demonstrate completeness, and soundness was demonstrated by extracting such strategies from proofs [1, 10, 20]. The *strategy extraction* procedures build partial strategies at each line of the proof, with the strategies at the final line forming a complete countermodel. These extraction procedures are based on the fact that in each application of a rule in the proof system, any winning strategies of the existential player are not destroyed.

In the systems QRes [25] and QURes [36], the soundness of the resolution rule is ensured by enforcing a very simple side-condition: variables other than the pivot cannot appear in both polarities in the antecedents. It was observed early on that this is often too restrictive. The *long-distance resolution proof system* LD-QRes [1, 38] arose from efforts to have less restrictive but still sound rules. In this system, a universal variable could appear in both polarities and get merged in the consequent, provided it was to the right of the pivot in the quantifier prefix. This preserves soundness, but the strategy extraction procedures become notably more complex.

The system LD-QRes, while provably better than QRes [20], is still needlessly restrictive in some situations. In particular, by checking a very simple syntactic prefix-ordering condition, it fails to exploit the fact that soundness is not lost even if universal variables to the left of the pivot are merged in both antecedents, provided the partial strategies built for them in both antecedents are identical. A *new system Merge Resolution (MRes)* was introduced last year [6] by a subset of the current authors, precisely to address this point. In MRes, partial strategies are explicitly represented within the proof, in a particular representation format called merge maps – these are essentially deterministic branching programs (DBPs). In this format, isomorphism checking can be done efficiently, and this opens the way for enabling sound applications of resolution that would have been blocked in LD-QRes (and QRes). In [6], it was shown that this brought a rich pay-off: there is a family of formulas, the SquaredEquality formulas, with short (linear-size) proofs in MRes, even in its tree-like and regular versions, but requiring exponential size in QRes, QURes, CP +  $\forall\text{Red}$ ,  $\forall\text{Exp} + \text{Res}$ , and IR. It is notable that the hardness of SquaredEquality in these systems stems from a certain semantic cost associated with these formulas and a corresponding lower bound [4, 5]. Thus the results of [6] show that such semantic costs are not a barrier for MRes.

In this paper, we explore the price paid for overcoming the semantic cost barrier. We show that (expectedly) MRes is not an unqualified success story. Building strategies into proofs via merge maps, and screening out unsoundness only through isomorphism tests, comes at a fairly heavy price.



**Figure 1** Visual summary of the proof complexity landscape, with new results shown in bold. Dotted lines to the box containing the four systems on the right indicate incomparability with all the four systems. All incomparability results with tree-like MRes hold also with the tree-like systems.

**(A) Lower bounds from circuit complexity for restricted versions of MRes.** Since the strategies are explicitly represented inside the proofs, computational hardness of strategies immediately translates to proof size lower bounds. While computational hardness of strategies is a known source of hardness in all reduction-based proof systems admitting efficient strategy extraction [8,10], the computational model relevant for MRes is one for which no unconditional lower bounds are known. For tree-like and regular MRes, the relevant models are decision trees and read-once DBPs, where lower bounds are known. Using this approach, we show:

1. Tree-like MRes is exponentially weaker than MRes.  
The QParity formulas witness the separation (Theorem 7) as their unique countermodel is the parity function which requires large decision trees.
2. Tree-like MRes is incomparable with the dag-like and tree-like versions of QRes, QURes, CP +  $\forall$ Red,  $\forall$ Exp + Res and IR.

One direction was shown in [6] via the SquaredEquality formulas: these formulas are easy for tree-like MRes but hard for dag-like QRes, QURes, CP +  $\forall$ Red,  $\forall$ Exp + Res, IR. The other direction is witnessed by the Completion Principle formulas (Theorem 9). Unlike the QParity formulas, these formulas do not have unique countermodels. However, we show that every countermodel requires large decision tree size, and hence obtain the lower bound for tree-like MRes.

**(B) Combinatorial lower bounds for full MRes.** Even when winning strategies are unique and easy to compute by DBPs, the formulas can be hard for MRes. We establish such hardness in two cases, obtaining more incomparabilities.

1. The LQParity formulas, easy in  $\forall\text{Exp} + \text{Res}$  [10], are exponentially hard for regular MRes (Theorem 13). Hence regular MRes is incomparable with  $\forall\text{Exp} + \text{Res}$  and IR.
2. The KBKF-lq formulas, easy in QURes [2], are exponentially hard for MRes (Theorem 19). Hence MRes and regular MRes are incomparable with QURes and  $\text{CP} + \forall\text{Red}$ .

The second hardness result above for the KBKF-lq formulas provides the first lower bound for the full system of MRes, for which previously no lower bounds were known.

It may be noted that for existentially quantified QBFs, all the QBF proof systems mentioned in this paper coincide with Resolution (or in case of  $\text{CP} + \forall\text{Red}$ , with Cutting Planes). Therefore lower bounds for these propositional proof systems trivially lift to the corresponding QBF proof system. In particular, the separations of tree-like and regular MRes from MRes and other systems follow from the propositional case. However, such lower bounds do not tell us much about the limitations of the QBF proof system other than what is known from the underlying propositional proof system. Therefore, in QBF proof complexity, we are interested in “genuine” QBF lower bounds, i.e. lower bounds that do not follow from propositional lower bounds (cf. [14] on how to formally define the notion of “genuine” lower bounds). The lower bounds we establish here are of this nature.

Figure 1 depicts the *simulation order and incomparabilities* we establish involving MRes and its refinements. Amongst the remaining systems (the five systems on the right), all relationships not directly implied by depicted connections are known to be incomparabilities [10, 12, 23].

## 2 Preliminaries

Let  $[n] = \{1, 2, \dots, n\}$  and  $[m, n] = \{m, \dots, n\}$ . We represent clauses by sets of literals.

The *resolution rule* derives, from clauses  $C \vee x$  and  $D \vee \neg x$ , the clause  $C \vee D$ . We say that  $C \vee D$  is the resolvent,  $x$  is the pivot, and denote this by  $C \vee D = \text{res}(C \vee x, D \vee \neg x, x)$ .

The *propositional proof system Resolution* proves that a CNF formula  $F$  is unsatisfiable by deriving the empty clause through repeated applications of the resolution rule.

**Quantified Boolean formulas.** A *Quantified Boolean formula* (QBF) in *prenex conjunctive normal form* is denoted  $\Phi := Q \cdot \phi$ , where (a)  $Q = Q_1 Z_1 Q_2 Z_2 \dots Q_k Z_k$  is the quantifier prefix, in which  $Z_i$  are pairwise disjoint finite sets of Boolean variables,  $Q_i \in \{\exists, \forall\}$  for each  $i \in [k]$  and  $Q_i \neq Q_{i+1}$  for each  $i \in [k-1]$ , and (b) the matrix  $\phi$  is a CNF over  $\text{vars}(\Phi) := \cup_{i \in [k]} Z_i$ .

The existential (resp. universal) variables of  $\Phi$ , typically denoted  $X$  or  $X_\exists$  (resp.  $U$  or  $X_\forall$ ) is the set obtained as a union of  $Z_i$  for which  $Q_i = \exists$  (resp.  $Q_i = \forall$ ). The prefix  $Q$  defines a binary relation  $<_Q$  on  $\text{vars}(\Phi)$ , such that  $z <_Q z'$  holds iff  $z \in Z_i$ ,  $z' \in Z_j$ , and  $i < j$ , in which case we say that  $z'$  is right of  $z$  and  $z$  is left of  $z'$ . For each  $u \in U$ , we define  $L_Q(u) := \{x \in X \mid x <_Q u\}$ , i.e. the existential variables left of  $u$ .

For a set of variables  $Z$ , let  $\langle Z \rangle$  denote the set of assignments to  $Z$ . A *strategy*  $h$  for a QBF  $\Phi$  is a set  $\{h^u \mid u \in U\}$  of functions  $h^u: \langle L_Q(u) \rangle \rightarrow \{0, 1\}$  (for each  $\alpha \in \langle X \rangle$ ,  $h^u(\alpha \upharpoonright_{L_Q(u)})$  and  $h(\alpha)$  should be interpreted as a Boolean assignment to the variable  $u$  and the variable set  $U$  respectively). Additionally  $h$  is *winning* if, for each  $\alpha \in \langle X \rangle$ , the restriction of  $\phi$  by the assignment  $(\alpha, h(\alpha))$  is false. We use the terms “winning strategy” and “countermodel” interchangeably. A QBF is called false if it has a countermodel, and true if it does not.

The semantics of QBFs is also explained by a *two-player evaluation game* played on a QBF. In a run of the game, two players, the existential and the universal player, assign values to the variables in the order of quantification in the prefix. The existential player wins if the assignment so constructed satisfies all the clauses of  $\phi$ ; otherwise the universal player wins. Assigning values according to a countermodel guarantees that the universal player wins no matter how the existential player plays; hence the term “winning strategy”.

## 2.1 The formulas

We describe the formulas we will use throughout the paper.

**The QParity and LQParity formulas [10].** Let  $\text{parity}^c(y_1, y_2, \dots, y_k)$  be a shorthand for the following conjunction of clauses:  $\bigwedge_{S \subseteq [k], |S| \equiv 1 \pmod{2}} ((\bigvee_{i \in S} \overline{y_i}) \vee (\bigvee_{i \notin S} y_i))$ . Thus  $\text{parity}^c(y_1, y_2, \dots, y_k)$  is equal to 1 iff  $y_1 + y_2 + \dots + y_k \equiv 0 \pmod{2}$ .  $\text{QParity}_n$  is the QBF  $\exists x_1, \dots, x_n, \forall z, \exists t_1, \dots, t_n. (\bigwedge_{i \in [n+1]} \phi_n^i)$  where

$$\phi_n^1 = \text{parity}^c(x_1, t_1); \quad \forall i \in [2, n], \phi_n^i = \text{parity}^c(t_{i-1}, x_i, t_i); \quad \phi_n^{n+1} = (t_n \vee z) \wedge (\overline{t_n} \vee \overline{z}).$$

The QBFs are false: they claim that there exist  $x_1, \dots, x_n$  such that  $x_1 + \dots + x_n$  is neither congruent to 0 nor 1 modulo 2. Note that the only winning strategy for the universal player is to play  $z$  satisfying  $z \equiv x_1 + \dots + x_n \pmod{2}$ .

Similarly, let  $\widehat{\text{parity}}^c(y_1, y_2, \dots, y_k, z)$  abbreviate  $\bigwedge_{C \in \text{parity}^c(y_1, y_2, \dots, y_k)} ((C \vee z) \wedge (C \vee \overline{z}))$ .  $\text{LQParity}_n$  is the QBF  $\exists x_1, \dots, x_n, \forall z, \exists t_1, \dots, t_n. (\bigwedge_{i \in [n+1]} \widehat{\phi}_n^i)$  where

$$\widehat{\phi}_n^1 = \widehat{\text{parity}}^c(x_1, t_1, z); \quad \forall i \in [2, n], \widehat{\phi}_n^i = \widehat{\text{parity}}^c(t_{i-1}, x_i, t_i, z); \quad \widehat{\phi}_n^{n+1} = (t_n \vee z) \wedge (\overline{t_n} \vee \overline{z}).$$

For both  $\text{QParity}_n$  and  $\text{LQParity}_n$ , for  $i, j \in [n+1], i \leq j$ , we let  $\phi_n^{[i,j]}$  denote  $\bigwedge_{k \in [i,j]} \phi_n^k$ . Also,  $X = \{x_1, \dots, x_n\}$  and  $T = \{t_1, \dots, t_n\}$ .

► **Observation 1.** For both  $\text{QParity}_n$  and  $\text{LQParity}_n$ : (a) for each  $i \in [n]$ , and each  $C \in \phi_n^i$ ,  $\{x_i, t_i\} \subseteq \text{var}(C)$ ; and (b) for each  $i \in [n+1] \setminus \{1\}$ , and each  $C \in \phi_n^i$ ,  $\{t_{i-1}\} \subseteq \text{var}(C)$ .

**The Completion Principle formulas  $\text{CR}_n$  [23].** The QBF  $\text{CR}_n$  is defined as follows:

$$\text{CR}_n = \exists_{i,j \in [n]} x_{ij}, \forall z, \exists_{i \in [n]} a_i, \exists_{j \in [n]} b_j. \left( \bigwedge_{i,j \in [n]} (A_{ij} \wedge B_{ij}) \right) \wedge L_A \wedge L_B$$

where  $A_{ij} = x_{ij} \vee z \vee a_i$ ,  $B_{ij} = \overline{x_{ij}} \vee \overline{z} \vee b_j$ ,  $L_A = \overline{a_1} \vee \dots \vee \overline{a_n}$ , and  $L_B = \overline{b_1} \vee \dots \vee \overline{b_n}$ . Let  $X, A, B$  denote the variable sets  $\{x_{ij} : i, j \in [n]\}$ ,  $\{a_i : i \in [n]\}$ , and  $\{b_j : j \in [n]\}$ . It is convenient to think of the  $X$  variables as arranged in an  $n \times n$  matrix.

Intuitively, the formulas describe a completion game, played on the matrix

$$\begin{pmatrix} a_1 & \dots & a_1 & \dots & a_n & \dots & a_n \\ b_1 & \dots & b_n & \dots & b_1 & \dots & b_n \end{pmatrix}$$

where the  $\exists$ -player first deletes exactly one cell per column and the  $\forall$ -player then chooses one row. The  $\forall$ -player wins if his row contains all of  $A$  or all of  $B$  (cf. [23]).

**The KBKF-lq[n] formulas [2].** Our last QBFs are a variant of the formulas introduced by Kleine Büning et al. [25], which in various versions appear prominently throughout the QBF literature [2, 5, 10, 20, 36]. For  $n > 1$ , the  $n$ th member of the KBKF-lq[n] family consists of the prefix  $\exists d_1, e_1, \forall x_1, \exists d_2, e_2, \forall x_2, \dots, \exists d_n, e_n, \forall x_n, \exists f_1, f_2, \dots, f_n$  and clauses

$$\begin{aligned} A_0 &= \{\overline{d_1}, \overline{e_1}, \overline{f_1}, \dots, \overline{f_n}\} \\ A_i^d &= \{d_i, x_i, \overline{d_{i+1}}, \overline{e_{i+1}}, \overline{f_1}, \dots, \overline{f_n}\} & A_i^e &= \{e_i, \overline{x_i}, \overline{d_{i+1}}, \overline{e_{i+1}}, \overline{f_1}, \dots, \overline{f_n}\} & \forall i \in [n-1] \\ A_n^d &= \{d_n, x_n, \overline{f_1}, \dots, \overline{f_n}\} & A_n^e &= \{e_n, \overline{x_n}, \overline{f_1}, \dots, \overline{f_n}\} \\ B_i^0 &= \{x_i, f_i, \overline{f_{i+1}}, \dots, \overline{f_n}\} & B_i^1 &= \{\overline{x_i}, f_i, \overline{f_{i+1}}, \dots, \overline{f_n}\} & \forall i \in [n-1] \\ B_n^0 &= \{x_n, f_n\} & B_n^1 &= \{\overline{x_n}, f_n\} \end{aligned}$$

## 12:6 Hard QBFs for Merge Resolution

Note that the existential part of each clause in KBKF-lq[n] is a Horn clause (at most one positive literal), and except  $A_0$ , is even strict Horn (exactly one positive literal).

We use the following shorthand notation. Sets of variables:  $D = \{d_1, \dots, d_n\}$ ,  $E = \{e_1, \dots, e_n\}$ ,  $F = \{f_1, \dots, f_n\}$ , and  $X = \{x_1, \dots, x_n\}$ . Sets of literals: For  $Y \in \{D, E, X, F\}$ , set  $Y^1 = \{u \mid u \in Y\}$  and  $Y^0 = \{\bar{u} \mid u \in Y\}$ . Sets of clauses:

$$\begin{aligned} \mathcal{A}_0 &= \{A_0\} \\ \mathcal{A}_i &= \{A_i^d, A_i^e\} \quad \forall i \in [n] & \mathcal{B}_i &= \{B_i^0, B_i^1\} \quad \forall i \in [n] \\ \mathcal{A}_{[i,j]} &= \bigcup_{k \in [i,j]} \mathcal{A}_k \quad \forall i, j \in [0, n], i \leq j & \mathcal{B}_{[i,j]} &= \bigcup_{k \in [i,j]} \mathcal{B}_k \quad \forall i, j \in [n], i \leq j \\ \mathcal{A} &= \mathcal{A}_{[0,n]} & \mathcal{B} &= \mathcal{B}_{[1,n]} \end{aligned}$$

We use the following property of these formulas:

► **Proposition 2.** *Let  $h$  be any countermodel for KBKF-lq[n]. Let  $\alpha$  be any assignment to  $D$ , and  $\beta$  be any assignment to  $E$ .*

*For each  $i \in [n]$ , if  $\alpha_j \neq \beta_j$  for all  $1 \leq j \leq i$ , then  $h^{x_i}((\alpha, \beta) \upharpoonright_{L_Q(x_i)}) = \alpha_i$ .*

*In particular, if  $\alpha_j \neq \beta_j$  for all  $j \in [n]$ , then the countermodel computes  $h(\alpha, \beta) = \alpha$ .*

## 2.2 The Merge Resolution proof system [6]

The formal definition of the *Merge Resolution proof system*, denoted MRes, is rather technical and can be found in [6]. Here we present a somewhat informal description.

First, we describe the *idea behind the proof system*. MRes is a line-based proof system. Each line  $L$  has a clause  $C$  with only existential literals, and a partial strategy  $h^u$  for each universal variable  $u$ . The idea is to maintain the invariant that for each existential assignment  $\alpha$ , if  $\alpha$  falsifies  $C$ , then  $\alpha$  extended by the partial universal assignment setting each  $u$  to  $h^u(\alpha)$  falsifies at least one of the clauses used to derive  $L$ . Thus the set of functions  $\{h^u\}$  gives a partial strategy that wins whenever the existential player plays from the set of assignments falsifying  $C$ . The goal is to derive a line with the empty clause; the corresponding strategy at that line will be a complete winning strategy, a countermodel. Along the way, resolution is used on the clauses. If the pivot is  $x$ , then for universal variables  $u$  right of  $x$ , the partial strategies can be combined with a branching decision on  $x$ . However, for  $u$  left of  $x$ , in the evaluation game, the value of  $u$  is already set when  $x$  is to be assigned. Thus already existing non-trivial partial strategies for  $u$  cannot be combined with a branching decision, and so this resolution step is blocked. However, if both the strategies are identical, or if one of them is trivial (unspecified), then the non-trivial strategy can be carried forward while maintaining the desired invariant. Checking whether strategies are identical can itself be hard, making verification of the proof difficult. In MRes, this is handled by choosing a particular representation called merge maps, where isomorphism checks are easy.

Now we can describe the proof system itself. First we describe *merge maps*. Syntactically, these are deterministic branching programs, specified by a sequence of instructions of one of the following two forms:

■  $\langle \text{line } \ell \rangle : b$  where  $b \in \{*, 0, 1\}$ .<sup>1</sup>

Merge maps containing a single such instruction are called simple. In particular, if  $b = *$ , then they are called trivial.

■  $\langle \text{line } \ell \rangle : \text{If } x = 0 \text{ then go to } \langle \text{line } \ell_1 \rangle \text{ else go to } \langle \text{line } \ell_2 \rangle$ , for some  $\ell_1, \ell_2 < \ell$ . In a merge map  $M$  for  $u$ , all queried variables  $x$  must precede  $u$  in the quantifier prefix.

Merge maps with such instructions are called complex.

<sup>1</sup> In [6], the notation used is  $b \in \{*, u, \bar{u}\}$ ;  $u, \bar{u}, *$  denote  $u = 1, u = 0$ , undefined respectively.

(All line numbers are natural numbers.) The merge map  $M^u$  computes a partial strategy for the universal variable  $u$  starting at the largest line number (the leading instruction) and following the instructions in the natural way. The value  $*$  denotes an undefined value.

Two merge maps  $M_1, M_2$  are said to be consistent, denoted  $M_1 \bowtie M_2$ , if for every line number  $i$  appearing in both  $M_1, M_2$ , the instructions with line number  $i$  are identical. Two merge maps  $M_1, M_2$  are said to be isomorphic, denoted  $M_1 \simeq M_2$ , if there is a bijection between the line numbers in  $M_1$  and  $M_2$  that transforms  $M_1$  to  $M_2$  in the natural way.

For the remainder of this section let  $\Phi = Q \cdot \phi$  be a QBF with existential variables  $X$  and universal variables  $U$ . The *proof system MRes* has the following rules:

1. *Axiom*: For a clause  $A$  in the matrix  $\phi$ , let  $C$  be the existential part of  $A$ . For each universal variable  $u$ , let  $b_u$  be the value  $u$  must take to falsify  $A$ ; if  $u \notin \text{var}(A)$ , then  $b_u = *$ . For any natural number  $i$ , the line  $(C, \{M^u : u \in U\})$  where each  $M^u$  is the simple merge map  $\langle i \rangle : b_u$  can be derived in MRes.
2. *Resolution*: From lines  $L_a = (C_a, \{M_a^u : u \in U\})$  for  $a \in \{0, 1\}$ , in MRes, the line  $L = (C, \{M^u : u \in U\})$  can be derived, where for some  $x \in X$ ,
  - $C = \text{res}(C_0, C_1, x)$ , and
  - for each  $u \in U$ , either  $M_a^u$  is trivial and  $M^u = M_{1-a}^u$  for some  $a$ , or  $M^u = M_0^u \simeq M_1^u$ , or  $x$  precedes  $u$  and  $M^u$  has a leading instruction that builds the complex merge map  $\text{If } x = 0 \text{ then } \langle M_0^u \rangle \text{ else } \langle M_1^u \rangle$ .

A *refutation* is a derivation using these rules and ending in a line with the empty existential clause. The size of the refutation is the number of lines. In the rest of this paper, we will denote refutations by the Greek letter  $\Pi$ .

A small but important illustrative example from [6] is reproduced in the appendix.

As shown in [6], the merge maps at the final line compute a countermodel for the QBF. To establish this, some stronger properties of the derivation are established and will be useful to us. We restate the relevant properties here.

► **Lemma 3** (Extracted/adapted from [6] Section 4.3, (Proof of Lemma 21)). *Let  $\Phi = Q \cdot \phi$  be a QBF with existential variables  $X$  and universal variables  $U$ . Let  $\Pi \stackrel{\text{def}}{=} L_1, \dots, L_m$  be an MRes refutation of  $\Phi$ , where each  $L_i = (C_i, \{M_i^u \mid u \in U\})$ . Further, for each  $i \in [m]$ ,*

- *let  $\alpha_i$  be the minimal partial assignment falsifying  $C_i$ ,*
- *let  $A_i$  be the set of assignments to  $X$  consistent with  $\alpha_i$ ,*
- *for each  $u \in U$ , let  $h_i^u$  be the function computed by  $M_i^u$ ,*
- *for each  $\alpha \in A_i$ , let  $h_i(\alpha)$  be the partial assignment which sets variable  $u$  to  $h_i^u(\alpha \upharpoonright_{L_Q(u)})$  if  $h_i^u(\alpha \upharpoonright_{L_Q(u)}) \neq *$ , and leaves it unset otherwise.*

*Then for each  $\alpha \in A_i$ , the assignment  $(\alpha, h_i(\alpha))$  falsifies at least one clause of  $\phi$  used in the sub-derivation of  $L_i$ .*

Let  $G_\Pi$  be the derivation graph corresponding to  $\Pi$  (with edges directed from the antecedents to the consequent, hence from the axioms to the final line).

► **Proposition 4** ([6]). *For all  $u \in U$ ,  $M_m^u$  is isomorphic to a subgraph of  $G_\Pi$  (up to path contraction).*

Let  $S$  be a subset of the existential variables  $X$  of  $\Phi$ . We say that an MRes refutation of  $\Phi$  is *S-regular* if for each  $x \in S$ , there is no leaf-to-root path that uses  $x$  as pivot more than once. An  $X$ -regular proof is simply called a *regular proof*. If  $G_\Pi$  is a tree, then we say that  $\Pi$  is a *tree-like proof*.

### 3 Lifting branching program lower bounds

The following lemma is an immediate consequence of Proposition 4.

► **Lemma 5.** *Let  $\Pi \stackrel{\text{def}}{=} L_1, \dots, L_m$  be an MRes refutation. If  $\Pi$  is tree-like (resp. regular), then for all  $u \in U$ ,  $M_m^u$  is a decision tree (resp. read-once branching program). Moreover, the size of  $\Pi$  is lower bounded by the size of  $M_m^u$ .*

This lemma allows us to lift lower bounds for decision trees (resp. read-once branching programs) to lower bounds for tree-like (resp. regular) Merge Resolution.

For  $\text{QParity}_n$  and  $\text{LQParity}_n$ , the only winning strategy for the universal player is to set  $z$  such that  $z \equiv x_1 + x_2 + \dots + x_n \pmod{2}$ .

► **Proposition 6 (Folklore).** *The decision tree size complexity of the parity function is  $2^n$ .*

► **Theorem 7.**  *$\text{size}_{\text{MResTree}}(\text{QParity}_n) = 2^{\Omega(n)}$  and  $\text{size}_{\text{MResTree}}(\text{LQParity}_n) = 2^{\Omega(n)}$ .*

For the QBF  $\text{CR}_n$ , the winning strategy for the universal player (countermodel) is not unique. However, we show that all countermodels require large decision trees.

► **Lemma 8.** *Every countermodel for  $\text{CR}_n$  has decision tree size complexity at least  $2^n$ .*

► **Theorem 9.**  *$\text{size}_{\text{MResTree}}(\text{CR}_n) = 2^{\Omega(n)}$ .*

► **Corollary 10.** *Tree-Like MRes is incomparable with the tree-like and general versions of QRes, QURes,  $\text{CP} + \forall\text{Red}$ ,  $\forall\text{Exp} + \text{Res}$ , and IR.*

**Proof.** We showed in Theorem 9 that the Completion Principle  $\text{CR}_n$  requires exponential-size refutations in tree-like Merge Resolution. It has polynomial-size refutations in tree-like QRes [22] (and hence also in QURes and  $\text{CP} + \forall\text{Red}$ ) and tree-like  $\forall\text{Exp} + \text{Res}$  [23] (and hence also in IR). (While [23] does not explicitly mention tree-like proofs, the proof provided there for  $\text{CR}_n$  is tree-like.) On the other hand, the formulas  $\text{EQ}_n$  have polynomial-size tree-like MRes refutations [6] but require exponential-size refutations in QRes, QURes,  $\text{CP} + \forall\text{Red}$  [5],  $\forall\text{Exp} + \text{Res}$ , IR [4] (cf. [3] on how to apply the lower bound technique from [4] to  $\text{EQ}_n$ ). ◀

We now show how to lift lower bounds for read-once branching programs to those for regular MRes. This follows the method used, for instance, in [10] (Section 4.1) and [30] (Section 6). Let  $f: X \rightarrow \{0, 1\}$  be a Boolean function, let  $C_f$  be a Boolean circuit encoding  $f$ , and let  $u$  be a variable not in  $X$ . Using Tseitin transformation [35], we can construct a CNF formula  $\phi(X, u, Y)$  such that  $\exists Y. \phi(X, u, Y)$  is logically equivalent to  $C_f(X) \neq u$ . Therefore,  $\Phi := \exists X \forall u \exists Y. \phi(X, u, Y)$ , called the QBF encoding of  $f$ , is a false QBF formula with  $f$  as the unique winning strategy. Moreover, the size of  $\Phi$  is polynomial in the size of  $C_f$ . Choosing a function  $f$  that can be computed by polynomial-size Boolean circuits but requires exponential-size read-once branching programs gives the desired lower bound. Many such functions are known [37]. For instance, we can use the following result:

► **Theorem 11 ([17]).** *There is a Boolean function  $f$  in  $n$  variables that can be computed by a Boolean circuit of size  $O(n^{3/2})$  but requires read-once branching programs of size  $2^{\Omega(\sqrt{n})}$ .*

► **Corollary 12.** *There is a Boolean function  $f$  in  $n$  variables with a QBF encoding  $\Phi$  of size polynomial in  $n$  such that any regular MRes refutation of  $\Phi$  has size  $2^{\Omega(\sqrt{n})}$ .*

#### 4 A lower bound for Regular Merge Resolution

In this section, we prove a lower for a formula whose countermodel can be computed by polynomial-size read-once branching programs.

► **Theorem 13.**  $\text{size}_{\text{MResReg}}(\text{LQParity}_n) = 2^{\Omega(n)}$ .

This follows from a stronger result that we prove below: any  $T$ -regular refutation of  $\text{LQParity}_n$  in MRes must have size  $2^{\Omega(n)}$  (Theorem 17).

The proof proceeds as follows: Let  $\Pi$  be a  $T$ -regular MRes refutation of  $\text{LQParity}_n$ . Since every axiom has a variable from  $T$  while the final clause in  $\Pi$  is empty, there is a maximal “component” of the proof leading to and including the final line, where all clauses are  $T$ -free. The clauses in this component involve only the  $X$  variables. We show that the “boundary” of this component is large, by showing in Lemma 16 that each clause here must be wide. (This idea was used in [30] to show that CR is hard for reductionless LD-QRes.) To establish the width bound, we note that no lines have trivial strategies. Since the pivots at the boundary are variables from  $T$ , the merge maps incoming into each boundary resolution must be isomorphic. By carefully analysing what axiom clauses can and must be used to derive lines just above the boundary (Lemma 15), we conclude that the merge maps must be simple, yielding the lower bound. To fill in all the details, we first describe some properties (Lemma 14) of  $\Pi$  that will be used in obtaining this result.

The lines of  $\Pi$  will be denoted by  $L, L', L''$  etc. For lines  $L$  and  $L'$  the respective clause, merge map and the function computed by the merge map will be denoted by  $C, M, h$  and  $C', M', h'$  respectively. Let  $G_\Pi$  be the derivation graph corresponding to  $\Pi$  (with edges directed from the antecedents to the consequent, hence from the axioms to the final line). We will refer to the nodes of this graph by the corresponding line. For  $L, L' \in \Pi$ , we will say  $L \rightsquigarrow L'$  if there is a path from  $L$  to  $L'$  in  $G_\Pi$ .

For a line  $L \in \Pi$ , let  $\Pi_L$  be the minimal sub-derivation of  $L$ , and let  $G_{\Pi_L}$  be the corresponding subgraph of  $G_\Pi$  with sink  $L$ . Define  $\text{UsedConstraints}(\Pi_L) = \{\phi_n^i \mid i \in [n+1], \text{leaves}(G_{\Pi_L}) \cap \phi_n^i \neq \emptyset\}$ , and  $\text{UCI}(\Pi_L) = \{i \in [n+1] \mid \phi_n^i \in \text{UsedConstraints}(\Pi_L)\}$ . (UCI stands for UsedConstraintsIndex.) Note that for any leaf  $L$ ,  $\text{UCI}(\Pi_L)$  is a singleton.

Define  $\mathcal{S}'$  to be the set of those lines in  $\Pi$  where the clause part has no  $T$  variable and furthermore there is a path in  $G_\Pi$  from the line to the final empty clause via lines where all the clauses also have no  $T$  variables. Let  $\mathcal{S}$  denote the set of leaves in the subgraph of  $G_\Pi$  restricted to  $\mathcal{S}'$ ; these are lines that are in  $\mathcal{S}'$  but their parents are not in  $\mathcal{S}'$ . Note that no leaf of  $\Pi$  is in  $\mathcal{S}'$  because all leaves of  $G_\Pi$  contain a variable in  $T$ .

► **Lemma 14.** *Let  $L = (C, M)$  be a line of  $\Pi$ . Then  $\text{UCI}(\Pi_L)$  is an interval  $[i, j]$  for some  $1 \leq i \leq j \leq n+1$ . Furthermore, (below  $i, j$  refer to the endpoints of this interval)*

1. For all  $k \in [i, j-1]$ ,  $t_k \notin \text{var}(C)$ .
2. If  $i > 1$ , then  $t_{i-1} \in \text{var}(C)$ .
3. If  $j \leq n$ , then  $t_j \in \text{var}(C)$ .
4.  $|\text{var}(C) \cap T| = 1$  iff  $[i, j]$  contains exactly one of  $1, n+1$ .  
 $\text{var}(C) \cap T = \emptyset$  iff  $[i, j] = [1, n+1]$ .
5. For all  $k \in [i, j] \cap [1, n]$ ,  $x_k \in \text{var}(C) \cup \text{var}(M)$ .

► **Lemma 15.** *Let  $L \in \mathcal{S}$  be derived in  $\Pi$  as  $L = \text{res}(L', L'', t_k)$ . Then  $\text{UCI}(\Pi_L) = [1, n+1]$ , and  $\text{UCI}(\Pi_{L'}), \text{UCI}(\Pi_{L''})$  partition  $[1, n+1]$  into  $[1, k], [k+1, n+1]$ .*

► **Lemma 16.** *For all  $L \in \mathcal{S}$ ,  $\text{width}(C) = n$ .*

## 12:10 Hard QBFs for Merge Resolution

► **Theorem 17.** *Every  $T$ -regular refutation of  $LQParity_n$  in  $MRes$  has size  $2^{\Omega(n)}$ .*

**Proof.** Let  $\Pi$  be a  $T$ -regular refutation of  $LQParity_n$  in  $MRes$ . Let  $\mathcal{S}', \mathcal{S}$  be as defined in the beginning of this sub-section. By definition, for each  $L = (C, M) \in \mathcal{S}'$ ,  $\text{var}(C) \subseteq X$ . Let  $\widehat{\Pi} = \{C \mid L = (C, M) \in \mathcal{S}'\}$ . Then  $\widehat{\Pi}$  contains a propositional resolution refutation of  $\mathcal{C} = \{C \mid L = (C, M) \in \mathcal{A}\}$ . Therefore  $\mathcal{C}$  is an unsatisfiable CNF formula over the  $n$  variables in  $X$ . By Lemma 16, each clause in  $\mathcal{C}$  has width  $n$  and so is falsified by exactly one assignment. Therefore, to ensure that each of the  $2^n$  assignments falsifies some clause, (at least)  $2^n$  clauses are required. Therefore  $|\mathcal{C}| \geq 2^n$ . Hence  $|\Pi| \geq 2^n$ . ◀

► **Corollary 18.** *Regular  $MRes$  is incomparable with  $\forall Exp + Res$  and  $IR$ .*

## 5 A lower bound for Merge Resolution

In this section we show that the KBKF-lq formulas are exponentially hard for  $MRes$ .

► **Theorem 19.**  $size_{MRes}(KBKF\text{-}lq[n]) = 2^{\Omega(n)}$ .

### Proof idea

We will show that, in any  $MRes$  refutation of the KBKF-lq formulas, the literals over the variables in  $F = \{f_1, f_2, \dots, f_n\}$  must be removed before the strategies become “very complex”. From this we conclude that there must be exponentially many lines.

To argue that literals over  $F$  must be removed before the strategies become “very complex”, we look at the form of the lines containing literals over  $F$ . If any such line has a “very complex” strategy (by which we mean that for some  $i \in [n]$ ,  $u_i$  depends on either  $d_i$  or  $e_i$ ), then the literals over  $F$  cannot be removed from the clause.

Elaborating on the roadmap of the argument: Let  $\Pi$  be an  $MRes$  refutation of  $KBKF\text{-}lq[n]$ . Each line in  $\Pi$  has the form  $L = (C, M^{x_1}, \dots, M^{x_n})$  where  $C$  is a clause over  $D, E, F$ , and each  $M^{x_i}$  is a merge map computing a strategy for  $x_i$ .

Define  $\mathcal{S}'$  to be set of those lines in  $\Pi$  where the clause part has no  $F$  variable and furthermore the line has a path in  $G_\Pi$  to the final empty clause via lines where all the clauses also have no  $F$  variables. Let  $\mathcal{S}$  denote the set of leaves in the subgraph of  $G_\Pi$  restricted to  $\mathcal{S}'$ ; these are lines that are in  $\mathcal{S}'$  but their parents are not in  $\mathcal{S}'$ . Note that by definition, for each  $L = (C, \{M^{x_i} \mid i \in [n]\}) \in \mathcal{S}'$ ,  $\text{var}(C) \subseteq D \cup E$ . No line in  $\mathcal{S}'$  (and in particular, no line in  $\mathcal{S}$ ) is an axiom since all axiom clauses have variables from  $F$ .

Recall that the variables of  $KBKF\text{-}lq[n]$  can be naturally grouped based on the quantifier prefix: for  $i \in [n]$ , the  $i$ th group has  $d_i, e_i, x_i$ , and the  $(n + 1)$ th group has the  $F$  variables. By construction, the merge map for  $x_i$  does not depend on variables in later groups, as is indeed required for a countermodel. We say that a merge map for  $x_i$  has self-dependence if it does depend on  $d_i$  and/or  $e_i$ .

We show that every merge map at every line in  $\mathcal{S}'$  is non-trivial (Lemma 24). Further, we show that at every line on the boundary of  $\mathcal{S}'$ , i.e. in  $\mathcal{S}$ , no merge map has self-dependence (Lemma 25). Using this, we conclude that  $\mathcal{S}$  must be exponentially large, since in every countermodel the strategy of each variable must have self-dependence (Proposition 2).

In order to show that lines in  $\mathcal{S}$  do not have self-dependence, we first establish several properties of the sets of axiom clauses used in a sub-derivation (Lemmas 20, 21, 22, 23).

**Detailed proof**

For a line  $L \in \Pi$ , let  $\Pi_L$  be the minimal sub-derivation of  $L$ , and let  $G_{\Pi_L}$  be the corresponding subgraph of  $G_\Pi$  with sink  $L$ . Let  $\text{UCI}(\Pi_L) = \{i \in [0, n] \mid \text{leaves}(G_{\Pi_L}) \cap \mathcal{A}_i \neq \emptyset\}$ . (UCI stands for UsedConstraintsIndex). Note that we are only looking at the clauses in  $\mathcal{A}$  to define UCI.

► **Lemma 20.** *For every line  $L = (C, \{M^{x_i} \mid i \in [n]\})$  of  $\Pi$ ,*

1.  $\text{UCI}(\Pi_L) = \emptyset$  if and only if  $C \cap F^1 \neq \emptyset$  if and only if  $|C \cap F^1| = 1$ .
2.  $\text{UCI}(\Pi_L) \neq \emptyset$  if and only if  $C \cap F^1 = \emptyset$ .

► **Lemma 21.** *A line  $L = (C, \{M^{x_i} \mid i \in [n]\})$  of  $\Pi$  with  $\text{UCI}(\Pi_L) = \emptyset$  has these properties:*

1.  $\text{var}(C) \subseteq F$ ; for all  $i \in [n]$ ,  $M^{x_i} \in \{*, 0, 1\}$ ;
2. For some  $j \in [n]$ ,  $f_j \in C$  and  $M^{x_j} \in \{0, 1\}$ ;
3. For  $1 \leq i < j$ ,  $f_i \notin \text{var}(C)$  and  $M^{x_i} = *$ ;
4. For  $j < i \leq n$ , if  $f_i \notin \text{var}(C)$ , then  $M^{x_j} \in \{0, 1\}$ .

► **Lemma 22.** *Let  $L = (C, \{M^{x_i} \mid i \in [n]\})$  be a line of  $\Pi$  with  $\text{UCI}(\Pi_L) \neq \emptyset$ . Then  $\text{UCI}(\Pi_L)$  is an interval  $[a, b]$  for some  $0 \leq a \leq b \leq n$ . Furthermore, (in the items below,  $a, b$  refer to the endpoints of this interval), it has the following properties:*

1. For  $k \in [n] \cap [a, b]$ ,  $M^{x_k} \neq *$ .
2. If  $a \geq 1$ , then  $|\overline{\{d_a, e_a\}} \cap C| = 1$ . If  $a = 0$ , then  $C$  does not have any positive literal.
3. If  $b < n$ , then  $\overline{d_{b+1}}, \overline{e_{b+1}} \in C$ .
4. For all  $k \in [n] \setminus [a, b]$ , (i)  $d_k, e_k \notin \text{var}(M^{x_k})$ , and (ii) if  $M^{x_k} = *$  then  $\overline{f_k} \in C$ .

► **Lemma 23.** *For any line  $L = (C, \{M^{x_i} \mid i \in [n]\})$  in  $\Pi$ , and any  $k \in [n]$ , if  $\{d_k, e_k\} \cap \text{var}(M^{x_k}) \neq \emptyset$ , then  $\text{UCI}(\Pi_L) = [a, n]$  for some  $a \leq k - 1$ .*

► **Lemma 24.** *For all  $L \in \mathcal{S}'$ , for all  $k \in [n]$ ,  $M^{x_k} \neq *$ .*

**Proof.** Consider a line  $L = (C, \{M^{x_i} \mid i \in [n]\}) \in \mathcal{S}'$ . Since  $L \in \mathcal{S}'$ ,  $\text{var}(C) \cap F = \emptyset$ , so  $C \cap F^1 = \emptyset$ . By Lemma 20,  $\text{UCI}(\Pi_L) \neq \emptyset$ . Since every clause in  $\mathcal{A}$  contains all literals in  $F^0$ , for each  $k \in [n]$ ,  $\Pi_L$  has a leaf where the clause contains  $\overline{f_k}$ . This literal is removed in deriving  $L$ , so  $\Pi_L$  also has a leaf where the clause contains the positive literal  $f_k$ . That is, it uses an axiom from  $\mathcal{B}_k$ ; this leaf has a non-trivial merge map for  $x_k$ . Since a step in MRes cannot make a non-trivial merge map trivial, the merge map for  $x_k$  at  $L$  is non-trivial. ◀

► **Lemma 25.** *For all  $L \in \mathcal{S}$ , for all  $k \in [n]$ ,  $d_k, e_k \notin \text{var}(M^{x_k})$ .*

**Proof.** Consider a line  $L \in \mathcal{S}$ ;  $L = (C, \{M^{x_i} \mid i \in [n]\})$ . Assume to the contrary that for some  $k \in [n]$ ,  $\{d_k, e_k\} \cap \text{var}(M^{x_k}) \neq \emptyset$ .

Line  $L$  is obtained by performing resolution on two non- $\mathcal{S}'$  clauses with a pivot from  $F$ . Let  $L = \text{res}(L', L'', f_\ell)$  for some  $\ell \in [n]$ ;  $f_\ell \in C'$  and  $\overline{f_\ell} \in C''$ . Since  $L$  has no variable in  $F$ ,  $f_\ell$  is the only variable from  $F$  in  $\text{var}(C')$  and  $\text{var}(C'')$ .

Since  $C'$  has the literal  $f_\ell \in F^1$ , by Observation 20,  $\text{UCI}(\Pi_{L'}) = \emptyset$  and  $L'$  is derived exclusively from  $\mathcal{B}$ . Since  $D \cup E$  and  $\text{var}(\mathcal{B})$  are disjoint, all the merge maps in  $L'$  have no variable from  $D \cup E$ . So  $M^{x_k}$  gets its  $D \cup E$  variables from  $(M'')^{x_k}$ . Since this does not block the resolution step,  $(M')^{x_k}$  must be trivial and  $M^{x_k} = (M'')^{x_k}$ . Since  $\text{var}(C') \cap F = f_\ell$ , by Lemma 21 (2),(3),(4),  $k < \ell$ .

The line  $L''$  has no literal from  $F^1$ , so by Observation 20,  $\text{UCI}(\Pi_{L''}) \neq \emptyset$ . It has a merge map for  $x_k$  involving at least one of  $d_k, e_k$ , so by Lemma 23,  $\text{UCI}(\Pi_{L''}) = [a, n]$  for some  $a \leq k - 1$ . Thus we have  $a \leq k - 1 < k < \ell \leq n$ .

Consider the resolution of  $L'$  with  $L''$ . By Lemma 21 (2),  $(M')^{x_\ell} \in \{0, 1\}$ , and by Lemma 22 (1),  $(M'')^{x_\ell} \neq *$ . To enable this resolution,  $(M'')^{x_\ell} = (M')^{x_\ell}$ . The clauses  $A_\ell^d$

## 12:12 Hard QBFs for Merge Resolution

and  $A_\ell^c$  give rise to different constant strategies for  $x_\ell$ . So the derivation of  $L''$  uses exactly one of these two clauses. Assume it uses  $A_\ell^d$ ; the other case is symmetric. Since  $a < \ell$ , the derivation of  $L''$  uses a clause from  $A_{\ell-1}$ , introducing literals  $\bar{d}_\ell$  and  $\bar{e}_\ell$ . Since the only clause containing positive literal  $e_\ell$  is not used,  $\bar{e}_\ell$  survives in  $C''$ . Going from  $L''$  to  $L$  removes only  $\bar{f}_\ell$ , so  $\bar{e}_\ell \in C$ .

To summarize, at this stage we know that  $L \in \mathcal{S}$ ,  $\bar{e}_\ell \in C$ ,  $\{d_k, e_k\} \cap \text{var}(M^{x_k}) \neq \emptyset$ ,  $M^{x_\ell} \in \{0, 1\}$  and  $1 \leq k < \ell \leq n$ .

Fix any path  $\rho$  in  $G_\Pi$  from  $L$  to  $L_\square$ . Along this path,  $e_\ell$  appears as the pivot somewhere, since the literal  $\bar{e}_\ell$  is eventually removed. Consider the resolution step at that point, say  $C_1 = \text{res}(C_2, C_3, e_\ell)$ , with  $C_3$  being the clause at the line on  $\rho$ . At the corresponding line  $L_3$ , the strategies are at least as complex as those at  $L$ . Hence  $\text{var}(M_3^{x_k}) \cap \{d_k, e_k\} \neq \emptyset$ . On the other hand,  $C_2$  has the positive literal  $e_\ell$ . By Lemma 22, for the corresponding line  $L_2$ ,  $\text{UCI}(\Pi_{L_2}) = [\ell, c]$  for some  $c \geq \ell$ . Since  $k < \ell$ , by Lemma 22,  $\{d_k, e_k\} \cap \text{var}(M_2^{x_k}) = \emptyset$ . However, the path from  $L_2$  to  $L_1$  and thence to  $L_\square$  along  $\rho$  witnesses that  $L_2 \in \mathcal{S}'$ , so by Lemma 24,  $(M_2)^{x_k} \neq *$ . Thus  $M_2^{x_k}$  and  $M_3^{x_k}$  are non-trivial but not isomorphic, and this blocks the resolution on  $e_\ell$ .

Thus our assumption that  $\{d_k, e_k\} \cap \text{var}(M^{x_k}) \neq \emptyset$  must be false. The lemma is proved.  $\blacktriangleleft$

**Proof.** (of Theorem 19) Let  $\Pi$  be a refutation of KBKF-lq[ $n$ ] in MRes. Let  $\mathcal{S}', \mathcal{S}$  be as defined in the beginning of this section. Let the final line of  $\Pi$  be  $L_\square = (\square, \{s^{x_i} \mid i \in [n]\})$ , and for  $i \in [n]$ , let  $h_i$  be the functions computed by the merge map  $s^{x_i}$ . By soundness of MRes, the functions  $\{h_i\}_{i \in [n]}$  form a countermodel for KBKF-lq[ $n$ ].

For each  $a \in \{0, 1\}^n$ , consider the assignment  $\alpha$  to the variables of  $D \cup E$  where  $d_i = a_i$ ,  $e_i = \bar{a}_i$ . Call such an assignment an anti-symmetric assignment. Given such an assignment, walk from  $L_\square$  towards the leaves of  $\Pi$  as far as is possible while maintaining the following invariant at each line  $L = (C, \{M^{x_i} \mid i \in [n]\})$  along the way:

1.  $\alpha$  falsifies  $C$ , and
2. for each  $i \in [n]$ ,  $h_i(\alpha) = M^{x_i}(\alpha)$ .

Clearly this invariant is initially true at  $L_\square$ , which is in  $\mathcal{S}'$ . If we are currently at a line  $L \in \mathcal{S}'$  where the invariant is true, and if  $L \notin \mathcal{S}$ , then  $L$  is obtained from lines  $L', L''$ . The resolution pivot in this step is not in  $F$ , since that would put  $L$  in  $\mathcal{S}$ . So both  $L'$  and  $L''$  are in  $\mathcal{S}'$ , and the pivot is in  $D \cup E$ . Let the pivot be in  $\{d_\ell, e_\ell\}$  for some  $\ell \in [n]$ . Depending on the pivot value, exactly one of  $C', C''$  is falsified by  $\alpha$ ; say  $C'$  is falsified. By Lemma 24, for each  $i \in [n]$ , both  $(M')^{x_i}$  and  $(M'')^{x_i}$  are non-trivial. By definition of the MRes rule,

- For  $i < \ell$ ,  $(M')^{x_i}$  and  $(M'')^{x_i}$  are isomorphic (otherwise the resolution is blocked), and  $M^{x_i} = (M')^{x_i} = (M'')^{x_i}$ .
- For  $i \geq \ell$ , there are two possibilities:
  - (1)  $(M')^{x_i}$  and  $(M'')^{x_i}$  are isomorphic, and  $M^{x_i} = (M')^{x_i}$ .
  - (2)  $M^{x_i}$  is a merge of  $(M')^{x_i}$  and  $(M'')^{x_i}$  with the pivot variable queried. By definition of the merge operation, since  $C'$  is falsified by  $\alpha$ ,  $M^{x_i}(\alpha) = (M')^{x_i}(\alpha)$ .

Thus in all cases, for each  $i$ ,  $h_i(\alpha) = M^{x_i}(\alpha) = (M')^{x_i}(\alpha)$ . Hence  $L'$  satisfies the invariant.

We have shown that as long as we have not encountered a line in  $\mathcal{S}$ , we can move further. We continue the walk until a line in  $\mathcal{S}$  is reached. We denote the line so reached by  $P(\alpha)$ . Thus  $P$  defines a map from anti-symmetric assignments to  $\mathcal{S}$ .

Suppose  $P(\alpha) = P(\beta) = (C, \{M^{x_i} \mid i \in [n]\})$  for two distinct anti-symmetric assignments obtained from  $a, b \in \{0, 1\}^n$  respectively. Let  $j$  be the least index in  $[n]$  where  $a_j \neq b_j$ . By Lemma 25,  $M^{x_j}$  depends only on  $\{d_i, e_i \mid i < j\}$ , and  $\alpha, \beta$  agree on these variables. Thus we get the equalities  $a_j = h_j(\alpha) = M^{x_j}(\alpha) = M^{x_j}(\beta) = h_j(\beta) = b_j$ , where the first and last

equalities follow from Proposition 2, the third equality from by Lemma 25 and choice of  $j$ , and the second and fourth equalities by the invariant satisfied at  $P(\alpha)$  and  $P(\beta)$  respectively. This contradicts  $a_j \neq b_j$ .

We have established that the map  $P$  is one-to-one. Hence,  $\mathcal{S}$  has at least as many lines as anti-symmetric assignments, so  $|\Pi| \geq |\mathcal{S}| \geq 2^n$ . ◀

► **Corollary 26.** *Both regular MRes and MRes are incomparable with QURes and CP+ $\forall$ Red.*

**Proof.** Theorem 19 shows that the KBKF-lq[ $n$ ] formula requires exponential-size refutations in MRes (and hence also in its regular restriction). It has polynomial-size refutations in QURes [2], and also in CP +  $\forall$ Red since CP +  $\forall$ Red simulates QURes ([12]). The other direction follows from the EQ $_n$  formulas, as already mentioned in the proofs of Corollaries 10, 18. ◀

## 6 Conclusions and Future Work

The proof system MRes was introduced in [6], using the novel idea of building strategies directly into the proof and using them to enable additional sound applications of resolution. In [6], the strengths of the proof system were demonstrated. In this paper, we complement that study by exposing some limitations of MRes. We obtain hardness for tree-like MRes by transferring computational hardness of the countermodels in decision trees, and for regular and general MRes by ad hoc combinatorial arguments.

Several questions still remain.

1. One of the driving goals behind the definition of MRes was overcoming a perceived weakness of LD-QRes: its criterion for blocking unsound applications of resolution also blocks several sound applications. However, whether MRes actually overcomes this weakness is yet to be demonstrated. In [6], MRes is shown to be more powerful than the reductionless variant of LD-QRes (introduced in [15] and further investigated in [6, 30]). However, we still do not have an instance of a formula hard for LD-QRes but easy for MRes. A natural candidate is LQParity, for which we only have a lower bound in regular MRes. Another natural candidate is SquaredEquality. The other direction, whether there is a formula easy for LD-QRes but hard for MRes, is also open. One possible candidate for this separation might appear to be KBKF, which is easy for LD-QRes [20] (that paper uses the name  $\varphi_t$ ). However the KBKF formulas can be shown to have short refutations in MRes as well, and hence cannot be used for this purpose.
2. In the propositional case, regular resolution simulates tree-like resolution. This relation may not hold in the case of MRes, and even if it does, it will need a different proof. The trick used in the propositional case – (i) interpret the proof tree as a decision tree for search, (ii) make the decision tree read-once, (iii) then return from the search tree to a refutation, – does not work here because when we prune away parts of the decision tree to get a read-once tree, we may end up destroying isomorphism of strategies of blocking variables.

---

## References

- 1 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Form. Methods Syst. Des.*, 41(1):45–65, August 2012.
- 2 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 154–169, Cham, 2014. Springer International Publishing.

- 3 Olaf Beyersdorff and Joshua Blinkhorn. Formulas with large weight: a new technique for genuine QBF lower bounds. *Electron. Colloquium Comput. Complex.*, 24:32, 2017.
- 4 Olaf Beyersdorff and Joshua Blinkhorn. Lower bound techniques for QBF expansion. *Theory of Computing Systems*, 64(3):400–421, 2020. doi:10.1007/s00224-019-09940-0.
- 5 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, Cost, and Capacity: A Semantic Technique for Hard Random QBFs. *Logical Methods in Computer Science*, Volume 15, Issue 1, February 2019. doi:10.23638/LMCS-15(1:13)2019.
- 6 Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building Strategies into QBF Proofs. *Journal of Automated Reasoning*, 2020. Preliminary version in 36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019). doi:10.1007/s10817-020-09560-1.
- 7 Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Hardness characterisations and size-width lower bounds for QBF resolution. In *Proc. ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 209–223. ACM, 2020.
- 8 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In *ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 249–260, 2016.
- 9 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Jan Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2), 2020.
- 10 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. New Resolution-Based QBF Calculi and Their Proof Complexity. *ACM Trans. Comput. Theory*, 11(4), September 2019. doi:10.1145/3352155.
- 11 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible interpolation for QBF resolution calculi. *Logical Methods in Computer Science*, 13, 2017.
- 12 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Understanding cutting planes for QBFs. *Information and Computation*, 262:141–161, 2018.
- 13 Olaf Beyersdorff, Leroy Chew, and Kartteek Sreenivasaiiah. A game characterisation of tree-like Q-Resolution size. *J. Comput. Syst. Sci.*, 104:82–101, 2019.
- 14 Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. *ACM Transactions on Computation Theory*, 12(2), 2020.
- 15 Nikolaj Bjørner, Mikoláš Janota, and William Klieber. On conflicts and strategies in QBF. In Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, and Andrei Voronkov, editors, *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning LPAR 2015*, volume 35 of *EPiC Series in Computing*, pages 28–41. EasyChair, 2015.
- 16 A. Blake. *Canonical expressions in boolean algebra*. PhD thesis, University of Chicago, 1937.
- 17 Beate Bollig and Ingo Wegener. A very simple function that requires exponential size read-once branching programs. *Information Processing Letters*, 66(2):53–57, 1998. doi:10.1016/S0020-0190(98)00042-8.
- 18 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.
- 19 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.
- 20 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19*, pages 291–308, 2013.
- 21 Marijn Heule, Martina Seidl, and Armin Biere. A unified proof system for QBF preprocessing. In *IJCAR*, pages 91–106, 2014.
- 22 Mikoláš Janota. On Q-Resolution and CDCL QBF solving. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 402–418, Cham, 2016. Springer International Publishing.
- 23 Mikoláš Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theoretical Computer Science*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

- 24 Manuel Kauers and Martina Seidl. Short proofs for some symmetric quantified Boolean formulas. *Inf. Process. Lett.*, 140:4–7, 2018.
- 25 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.
- 26 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.
- 27 Florian Lonsing, Uwe Egly, and Martina Seidl. Q-resolution with generalized axioms. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 435–452. Springer, 2016.
- 28 Jakob Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015.
- 29 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long-distance Q-resolution with dependency schemes. *J. Autom. Reasoning*, 63(1):127–155, 2019.
- 30 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Proof complexity of fragments of long-distance Q-resolution. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 319–335. Springer, 2019.
- 31 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (QBF EVAL’16 and QBF EVAL’17). *Artif. Intell.*, 274:224–248, 2019.
- 32 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.
- 33 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019*, pages 78–84, 2019.
- 34 Friedrich Slivovsky and Stefan Szeider. Soundness of Q-resolution with dependency schemes. *Theoretical Computer Science*, 612:83–101, 2016.
- 35 G. S. Tseitin. On the complexity of derivation in propositional calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1\_28.
- 36 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Proc. Principles and Practice of Constraint Programming (CP’12)*, pages 647–663, 2012.
- 37 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719789.
- 38 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002*, pages 442–449, 2002.



# On Sampling Based Algorithms for $k$ -Means

**Anup Bhattacharya**

Indian Statistical Institute Kolkata, India  
bhattacharya.anup@gmail.com

**Dishant Goyal**

Indian Institute of Technology Delhi, India  
Dishant.Goyal@cse.iitd.ac.in

**Ragesh Jaiswal**<sup>1</sup>

Indian Institute of Technology Delhi, India  
rjaiswal@cse.iitd.ac.in

**Amit Kumar**

Indian Institute of Technology Delhi, India  
amitk@cse.iitd.ac.in

---

## Abstract

---

We generalise the results of Bhattacharya et al.[9] for the *list- $k$ -means* problem defined as – for a (unknown) partition  $X_1, \dots, X_k$  of the dataset  $X \subseteq \mathbb{R}^d$ , find a *list* of  $k$ -center-sets (each element in the list is a set of  $k$  centers) such that at least one of  $k$ -center-sets  $\{c_1, \dots, c_k\}$  in the list gives an  $(1 + \varepsilon)$ -approximation with respect to the cost function  $\min_{\text{permutation } \pi} \left[ \sum_{i=1}^k \sum_{x \in X_i} \|x - c_{\pi(i)}\|^2 \right]$ . The list- $k$ -means problem is important for the constrained  $k$ -means problem since algorithms for the former can be converted to PTAS for various versions of the latter. The algorithm for the list- $k$ -means problem by Bhattacharya et al. is a  $D^2$ -sampling based algorithm that runs in  $k$  iterations. Making use of a constant factor solution for the (classical or unconstrained)  $k$ -means problem, we generalise the algorithm of Bhattacharya et al. in two ways – (i) for any fixed set  $X_{j_1}, \dots, X_{j_t}$  of  $t \leq k$  clusters, the algorithm produces a list of  $\binom{k}{\varepsilon}^{O(\frac{1}{\varepsilon})}$   $t$ -center sets such that (w.h.p.) at least one of them is good for  $X_{j_1}, \dots, X_{j_t}$ , and (ii) the algorithm runs in a single iteration. Following are the consequences of our generalisations:

1. *Faster PTAS under stability and a parameterised reduction*: Property (i) of our generalisation is useful in scenarios where finding good centers becomes easier once good centers for a few “bad” clusters have been chosen. One such case is clustering under stability of Awasthi et al.[5] where the number of such bad clusters is a constant. Using property (i), we significantly improve the running time of their algorithm from  $O(dn^3)(k \log n)^{\text{poly}(\frac{1}{\beta}, \frac{1}{\varepsilon})}$  to  $O\left(dn^3 \left(\frac{k}{\varepsilon}\right)^{O(\frac{1}{\beta\varepsilon^2})}\right)$ . Another application is a parameterised reduction from the *outlier* version of  $k$ -means to the classical one where the bad clusters are the outliers.
2. *Streaming algorithms*: The sampling algorithm running in a single iteration (i.e., property (ii)) allows us to design a constant-pass, logspace streaming algorithm for the list- $k$ -means problem. This can be converted to a constant-pass, logspace streaming PTAS for various constrained versions of the  $k$ -means problem. In particular, this gives a 3-pass, polylog-space streaming PTAS for the constrained binary  $k$ -means problem which in turn gives a 4-pass, polylog-space streaming PTAS for the generalised binary  $\ell_0$ -rank- $r$  approximation problem. This is the first constant pass, polylog-space streaming algorithm for either of the two problems. *Coreset* based techniques, which is another approach for designing streaming algorithms in general, is not known to work for the constrained binary  $k$ -means problem to the best of our knowledge.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Facility location and clustering

**Keywords and phrases**  $k$ -means, low rank approximation

---

<sup>1</sup> Part of this work was done while the author was on a sabbatical from IIT Delhi and visiting UC San Diego.



Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.13

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/1909.07511> and <https://arxiv.org/abs/1909.11744>.

**Acknowledgements** The authors would like to thank Sanjeev Khanna and Sepehr Assadi for allowing us to use their impossibility argument for the chromatic  $k$ -means problem. Anup Bhattacharya would like to thank SERB-National Post Doctoral Fellowship, India. Dishant Goyal would like to thank TCS Research Scholar Program.

## 1 Introduction

Clustering is one of the most important tools for data analysis and the  $k$ -means clustering problem is one of the most prominent mathematical formulations of clustering. The goal of clustering is to partition data objects into groups, called *clusters*, such that similar objects are in the same cluster and dissimilar ones are in different clusters. Defining the clustering problem formally requires us to quantify the notion of similarity/dissimilarity and there are various ways of doing this. Given that in most contexts data objects can be represented as vectors in  $\mathbb{R}^d$ , a natural notion of distance between data points is the squared Euclidean distance and this gives rise to the  $k$ -means problem.

**$k$ -means:** Given a dataset  $X \subset \mathbb{R}^d$  and a positive integer  $k$ , find a set  $C \subset \mathbb{R}^d$  of  $k$  points, called *centers*, such that the following cost function is minimised:  $\Phi(C, X) \equiv \sum_{x \in X} \min_{c \in C} \|x - c\|^2$ .<sup>2</sup>

The  $k$ -means problem has been widely studied by both theoreticians and practitioners and is quite uniquely placed in the computer science research literature. The theoretical worst-case analysis properties of the  $k$ -means problem are fairly well understood. The problem is known to be NP-hard [17, 36, 39] and APX-hard [6, 14]. A lot of work has been done on obtaining efficient constant approximation algorithms for this problem (e.g., [30, 2]). However, this is not the main focus of this work. In this work, we discuss approximation schemes for the  $k$ -means problem and its variants. Approximation schemes are a family of algorithms  $\{A\}_\varepsilon$  that give  $(1 + \varepsilon)$ -approximation guarantee.

Given the hardness of approximation results, it is known that a *Polynomial Time Approximation Scheme (PTAS)* is not possible unless  $P = NP$ . However, there are efficient approximation schemes when at least one of  $k, d$  is not part of the input (and hence assumed to be a fixed constant). The work on approximation schemes for the  $k$ -means problem can be split into two categories where one consists of algorithms under the assumption that  $k$  is a constant while the other with  $d$  as a constant. Assuming  $k$  is a constant, there are various PTAS [32, 19, 28, 29] with running time  $O(nd \cdot 2^{\tilde{O}(\frac{k}{\varepsilon})})$ .<sup>3</sup> Note that the running time has a dependence on  $2^k$ . This is nicely supported by a conditional lower bound result [3] that says that under the *Exponential Time Hypothesis (ETH)* any approximation algorithm (beyond a fixed approximation factor) that runs in time polynomial in  $n$  and  $d$  will have a running time dependence of at least  $2^k$ . On the other hand, PTAS based on the assumption that  $d$  is a constant form another line of research culminating in the work of Addad et al. [15] and Friggstad et al. [23] who gave a local search based PTAS with running time dependence on  $d$

<sup>2</sup> For a singleton set  $C = \{c\}$ , we will use  $\Phi(c, X)$  and  $\Phi(\{c\}, X)$  interchangeably.

<sup>3</sup> The multiplicative factor of  $nd$  can be changed to an additive factor using useful data analysis tools and techniques such as *coresets* [19] and *dimensionality reduction* [34].

of the form  $(\frac{k}{\varepsilon})^\zeta$  where  $\zeta = \frac{d^{O(d)}}{\varepsilon^{O(\frac{d}{\varepsilon})}}$ . The work of Makarychev et al. [37] nicely consolidates the two lines of work by showing that the cost of the optimal  $k$ -means solution is preserved up to a factor of  $(1 + \varepsilon)$  under a projection onto a random  $O\left(\frac{\log(k/\varepsilon)}{\varepsilon^2}\right)$ -dimensional subspace.

The  $k$ -means problem nicely models the *locality* requirement of clustering. That is, similar (or closely located points) should be in the same cluster and dissimilar (or far-away points) should be in different clusters. However, in many different clustering contexts in machine learning and data mining, locality is not the only desired clustering property. There are other constraints in addition to the locality requirement. For example, one requirement is that the clusters should be balanced or in other words contain roughly equal number of points. Modelling such requirements within the framework of the  $k$ -means problem gives rise to the so-called *constrained  $k$ -means problem*. The constrained  $k$ -means problem can be modelled as follows: Let  $\mathbb{C}$  denote the set of  $k$ -clusterings that satisfy the relevant constraint. Then the goal is to find a clustering  $\mathcal{X} = \{X_1, \dots, X_k\}$  of the dataset  $X \subset \mathbb{R}^d$  such that the clustering  $\mathcal{X}$  belongs to  $\mathbb{C}$  and the following cost function is minimised:

$$\Delta(\mathcal{X}) \equiv \sum_{i=1}^k \Delta(X_i), \text{ where } \Delta(X_i) \equiv \Phi(\mu(X_i), X_i) \text{ and } \mu(X_i) \equiv \frac{\sum_{x \in X_i} x}{|X_i|}.$$

Note that  $\mu(X_i)$  is the *centroid* of the data points  $X_i$ . It can be easily shown that the centroid gives the best 1-means cost for any dataset and so  $\Delta(X_i)$  denotes the optimal 1-means cost of dataset  $X_i$ . The above formulation in terms of the feasible clusterings  $\mathbb{C}$  is an attempt to give a unified framework for considering different variations of the constrained clustering problem. The issue with such an attempt is how to concisely represent the set of feasible clusterings  $\mathbb{C}$ . This issue was addressed in the nice work of Ding and Xu [18] who gave a unified framework for considering constrained versions of the  $k$ -means problem. For every constrained version, instead of defining  $\mathbb{C}$  they define a *partition algorithm*  $\mathcal{P}^{\mathbb{C}}$  which, when given a set of  $k$  centers  $\{c_1, \dots, c_k\}$ , outputs a feasible clustering  $\{X_1, \dots, X_k\}$  (i.e., a clustering in  $\mathbb{C}$ ) that minimises the cost  $\sum_{i=1}^k \Phi(\{c_i\}, X_i)$ . They give efficient partition algorithms for a variety of constrained  $k$ -means problems. These problems and their description are given in Table 1. Note that the partition algorithm for the  $k$ -means problem (i.e., the classical unconstrained version) is simply the *Voronoi partitioning* algorithm.

Efficient partition algorithms allow us to design PTAS in the following manner: Let  $\mathcal{X} = \{X_1, \dots, X_k\}$  be an optimal clustering for some constrained  $k$ -means problem with optimal cost  $OPT = \Delta(\mathcal{X}) = \sum_{i=1}^k \Delta(X_i)$ . Suppose in some way, we are able to find a  $k$ -center-set  $\{c_1, \dots, c_k\}$  such that  $\min_{\text{permutation } \pi} \left[ \sum_{i=1}^k \sum_{x \in X_i} \|x - c_{\pi(i)}\|^2 \right] \leq (1 + \varepsilon) \cdot OPT$ . Then we can use the partition algorithm to find a clustering  $\bar{\mathcal{X}} = \{\bar{X}_1, \dots, \bar{X}_k\}$  such that  $\Delta(\bar{\mathcal{X}}) \leq (1 + \varepsilon) \cdot OPT$ . It turns out that even though producing a single such  $k$ -center-set may not be possible, producing a *list* of such  $k$ -center-sets is possible. Using the partition algorithm to find the clustering with least cost from the list will give us a  $(1 + \varepsilon)$ -approximate solution. This is the main idea used for designing PTAS by Ding and Xu [18] and Bhattacharya et al. [9]. Bhattacharya et al. [9] gave quantitative improvements over the results of Ding and Xu in terms of the list size. They also formally defined the *list- $k$ -means* problem that is a natural problem in the context of the above discussion.<sup>4</sup> One of the main focus of discussion of this paper will be the list- $k$ -means problem. So, let us first define the problem formally.

<sup>4</sup> Note that Ding and Xu [18] implicitly gave an algorithm for list- $k$ -means without naming it so.

■ **Table 1** Constrained  $k$ -means problems with efficient partition algorithm (see Section 4 in [18]).

#	Problem	Description
1.	$r$ -gather $k$ -means clustering ( $r, k$ )-GMeans	Find clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ with minimum $\Delta(\mathcal{X})$ such that for all $i$ , $ X_i  \geq r$
2.	$r$ -Capacity $k$ -means clustering ( $r, k$ )-CaMeans	Find clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ with minimum $\Delta(\mathcal{X})$ such that for all $i$ , $ X_i  \leq r$
3.	$l$ -Diversity $k$ -means clustering ( $l, k$ )-DMeans	Given that every data point has an associated colour, find a clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ with minimum $\Delta(\mathcal{X})$ such that for all $i$ , the fraction of points sharing the same colour inside $X_i$ is $\leq \frac{1}{l}$
4.	Chromatic $k$ -means clustering $k$ -ChMeans	Given that every data point has an associated colour, find a clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ with minimum $\Delta(\mathcal{X})$ such that for all $i$ , $X_i$ should not have more than one point with the same colour.
5.	Fault tolerant $k$ -means clustering ( $l, k$ )-FMeans	Find clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ such that the sum of squared distances of the points to the $l$ nearest centers out of $\{\mu(X_1), \dots, \mu(X_k)\}$ , is minimised.
6.	Semi-supervised $k$ -means clustering $k$ -SMeans	Given a target clustering $\mathcal{X}' = \{X'_1, \dots, X'_k\}$ and constant $\alpha$ find a clustering $\mathcal{X} = \{X_1, \dots, X_k\}$ such that the cost $\alpha \cdot \Delta(\mathcal{X}) + (1 - \alpha) \cdot \text{Dist}(\mathcal{X}', \mathcal{X})$ is minimised. $\text{Dist}$ denotes the set-difference distance.

**List- $k$ -means:** Let  $X \subset \mathbb{R}^d$  be the dataset and let  $\mathcal{X} = \{X_1, \dots, X_k\}$  be an arbitrary clustering of dataset  $X$ . Given  $X$ , positive integer  $k$ , and error parameter  $\varepsilon > 0$ , find a *list* of  $k$ -center-sets such that (w.h.p.<sup>5</sup>) at least one of the sets gives  $(1 + \varepsilon)$ -approximation with respect to the cost function:  $\Psi(\{c_1, \dots, c_k\}, \mathcal{X}) \equiv \min_{\text{permutation } \pi} \left[ \sum_{i=1}^k \sum_{x \in X_i} \|x - c_{\pi(i)}\|^2 \right]$ .

Bhattacharya et al. [9] gave a lower bound on the list size using a counting argument and a closely matching upper bound using a  $D^2$ -sampling based approach.  $D^2$ -sampling is a simple idea that is very useful in the context of the  $k$ -means/median clustering problems. Here, the centers are sampled from the given dataset in successive iterations where the probability of a point getting sampled as the center in an iteration is proportional to the squared distance of this point to the nearest center out of the centers already chosen in the previous iterations. Before discussing the algorithm for the list- $k$ -means problem, let us first make sure that the relevance of this problem in the context of the constrained  $k$ -means problems is well understood. Indeed, given any constrained  $k$ -means clustering problem with feasible clusterings  $\mathbb{C}$  and partition algorithm  $\mathcal{P}^{\mathbb{C}}$ , one can obtain a  $(1 + \varepsilon)$ -approximate solution by first running an algorithm for the list  $k$ -means problem (where the unknown clustering is any optimal clustering for the constrained  $k$ -means problem) to obtain a list  $\mathcal{L}$  and then use the partition algorithm  $\mathcal{P}^{\mathbb{C}}$  to pick the minimum cost clustering from  $\mathcal{L}$ . From the previous discussion, it should be clear that this will give us a  $(1 + \varepsilon)$ -approximate solution (w.h.p.). Let us now discuss the  $D^2$ -sampling based algorithm for the list- $k$ -means problem.

Bhattacharya et al. [9] gave an algorithm for the list- $k$ -means problem with list size  $|\mathcal{L}| = \left(\frac{k}{\varepsilon}\right)^{O\left(\frac{k}{\varepsilon}\right)}$  and running time  $O(nd|\mathcal{L}|)$ . Their algorithm explores a rooted tree of size  $\left(\frac{k}{\varepsilon}\right)^{O\left(\frac{k}{\varepsilon}\right)}$  and depth  $k$  where the degree of every non-leaf vertex is  $\left(\frac{k}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon}\right)}$ . Every node in this tree has an associated center and the path from root to a leaf node gives one of the  $k$ -center-sets for the output list. Let  $v$  be an internal node at depth  $i$ . The path from root to  $v$  defines  $i$  centers  $C_v$  and their algorithm extends these  $i$  centers to  $(i + 1)$  centers by  $D^2$ -sampling  $\text{poly}\left(\frac{k}{\varepsilon}\right)$  points w.r.t.  $C_v$  and considering the centroids of all possible subsets

<sup>5</sup> We use w.h.p. as an abbreviation for “with high probability”.

of size  $O(\frac{1}{\varepsilon})$  of the sampled points plus copies of centers in  $C_v$ .<sup>6</sup> This defines the  $(\frac{k}{\varepsilon})^{O(\frac{1}{\varepsilon})}$  children of  $v$  that are further explored subsequently. In their analysis, they showed that for every node  $v$ , there is always (w.h.p.) a child of  $v$  that is a good center for one of the clusters for which none of the centers in  $C_v$  is good.

Note that the algorithm of Bhattacharya et al.[9] in the previous paragraph has an unavoidable iteration of depth  $k$  since their analysis works only when the centers are picked *one-by-one* in  $k$  iterations. We circumvent this inherent restriction by using a constant factor approximate solution  $C$  to the  $k$ -means problem (i.e., the unconstrained  $k$ -means problem) for the given dataset  $X$ . That is,  $\Phi(C, X) \leq \alpha \cdot OPT^*$ , where  $OPT^*$  denotes the optimal  $k$ -means cost. Note that there are a number of constant factor approximation algorithms available for the  $k$ -means problem. So, this assumption is not restrictive at all. We can even further relax the assumption by noting that an  $(O(1), O(1))$  bi-criteria approximate solution  $C$  is sufficient. This means that  $|C| = O(k)$  and  $\Phi(C, X) \leq \alpha \cdot OPT^*$ . There are bi-criteria approximation algorithms available for the  $k$ -means problem. For example, there is a simple  $O(nkd)$  bi-criteria approximation algorithm based on  $D^2$ -sampling that just samples  $O(k)$  points (using  $D^2$ -sampling) and it has been shown [1] that the set of centers obtained gives a constant approximation with high probability. Making use of a constant factor solution  $C$ , we generalise the  $D^2$ -sampling based algorithm of Bhattacharya et al. [9] in two ways:

1. We consider the case where we may not need to find good centers for *all* clusters but for  $t \leq k$  clusters  $X_{j_1}, \dots, X_{j_t}$ . For any fixed choice of  $t$  clusters  $X_{j_1}, \dots, X_{j_t}$ , our algorithm returns a list of  $(\frac{k}{\varepsilon})^{O(\frac{t}{\varepsilon})}$   $t$ -center sets such that (w.h.p.) at least one of them is “good” for  $X_{j_1}, \dots, X_{j_t}$ . Note that the list size is exponential in  $t$  but not in  $k$ . This can be useful in scenarios where finding good centers of most of the clusters becomes easier (or not even required) once good centers of a few  $t \ll k$  clusters have been chosen.
2. The sampling algorithm runs in a *single* iteration where  $\text{poly}(\frac{t}{\varepsilon})$  points from  $X$  are  $D^2$ -sampled w.r.t.  $C$ . We show that good centers for clusters  $X_{j_1}, \dots, X_{j_t}$  can simultaneously be found from the sampled points and points in the set  $C$ . (Note that there is an iteration for probability amplification in algorithm **GoodCenters** but since the  $2^t$  rounds are independent, they can be executed independently.)

The formal description of the generalised algorithm is given in Algorithm 1. The algorithm takes as input dataset  $X$ , an  $\alpha$ -approximate solution  $C$ , error parameter  $\varepsilon$ , and  $t$  and outputs a list  $\mathcal{L}$  of  $t$ -center sets. Note that the list size produced by the above algorithm is  $|\mathcal{L}| = (\frac{k}{\varepsilon})^{O(\frac{t}{\varepsilon})}$  and running time is  $O(nd|\mathcal{L}|)$ . We will show that the **GoodCenters** algorithm behaves well (w.h.p.) for **any** fixed set of  $t$  clusters  $X_{j_1}, \dots, X_{j_t}$  out of clusters  $X_1, \dots, X_k$ . What this means is that for **any** fixed set of  $t$  clusters  $X_{j_1}, \dots, X_{j_t}$ , the list  $\mathcal{L}$  produced by the **GoodCenters** algorithm will (w.h.p.) contain a  $t$ -center set  $\mathcal{C}$  that is good for these clusters  $X_{j_1}, \dots, X_{j_t}$ . This is our main result on list- $k$ -means and we formally state this as the next theorem.

► **Theorem 1 (Main Theorem).** *Let  $0 < \varepsilon \leq \frac{1}{2}$  and  $t$  be any positive integer. Let  $X_{j_1}, \dots, X_{j_t}$  denote an arbitrary set of  $t$  clusters out of  $k$  clusters  $X_1, \dots, X_k$  of the dataset  $X$ . Let  $\mathcal{L}$  denote the list returned by the algorithm **GoodCenters**( $X, C, \varepsilon, t$ ). Then with probability at least  $\frac{3}{4}$ ,  $\mathcal{L}$  contains a center set  $\mathcal{C}$  such that:*

$$\Psi(\mathcal{C}, \{X_{j_1}, \dots, X_{j_t}\}) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \sum_{i=1}^t \Delta(X_{j_i}) + \frac{\varepsilon}{2} \cdot OPT \leq (1 + \varepsilon) \cdot OPT, \quad \text{where } OPT = \sum_{i=1}^k \Delta(X_i).$$

<sup>6</sup>  $D^2$ -sampling w.r.t. a center set  $C$  implies sampling from the dataset  $X$  using a distribution where the probability of sampling point  $x$  is proportional to  $\min_{c \in C} \|x - c\|^2$ . In the case  $C = \emptyset$ ,  $D^2$ -sampling is the same as uniform sampling.

■ **Algorithm 1** Algorithm for finding good centers.

---

GoodCenters( $X, C, \varepsilon, t$ )

**Inputs:** Dataset  $X$ ,  $\alpha$ -approximate  $C$ , accuracy  $\varepsilon$ , and number of centers  $t$

**Output:** A list  $\mathcal{L}$ , each element in  $\mathcal{L}$  being a  $t$ -center set

**Constants:**  $\eta = \frac{2^{16}\alpha t}{\varepsilon^4}$ ;  $\tau = \frac{128}{\varepsilon}$

(1)  $\mathcal{L} \leftarrow \emptyset$

(2) Repeat  $2^t$  times:

(3) Sample a multi-set  $M$  of  $\eta t$  points from  $X$  using  $D^2$ -sampling wrt center set  $C$

(4)  $M \leftarrow M \cup \{\frac{128t}{\varepsilon} \text{ copies of each element in } C\}$

(5) For all disjoint subsets  $S_1, \dots, S_t$  of  $M$  such that  $\forall i, |S_i| = \tau$ :

(6)  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mu(S_1), \dots, \mu(S_t))\}$

(7) return( $\mathcal{L}$ )

---

Moreover,  $|\mathcal{L}| = (\frac{k}{\varepsilon})^{O(\frac{t}{\varepsilon})}$  and the running time of the algorithm is  $O(nd|\mathcal{L}|)$ .

We shall formally prove the above theorem in the full version of the paper. We give a high-level discussion here. Without loss of generality, we will assume that  $j_i = i$ , that is the  $t$  clusters  $X_{j_1}, \dots, X_{j_t}$  are the first  $t$  clusters  $X_1, \dots, X_t$ . Since, the input center set  $C$  is an  $(\alpha, \beta)$ -approximate solution to the standard  $k$ -means problem on dataset  $X$ , we have

$$\Phi(C, X) \leq \alpha \cdot OPT^* \quad \text{and} \quad |C| \leq \beta k \quad (1)$$

Note that the outer iteration (repeat  $2^t$  times in line (2)) is to amplify the probability that the list  $\mathcal{L}$  containing a good  $t$ -center set. We will show that the probability of finding a good  $t$ -center set in one iteration is at least  $(3/4)^t$  and the theorem follows from simple probability calculation. So in the remaining discussion we will only discuss one iteration of the algorithm. Consider the multi-set  $M$  after line (3) of the algorithm. We will show that with probability at least  $(3/4)^t$ , there are disjoint (multi) subsets  $T_1, \dots, T_t$  each of size  $\tau$  such that for every  $j = 1, \dots, t$ ,

$$\Phi(\mu(T_j), X_j) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(X_j) + \frac{\varepsilon}{2t} \cdot OPT. \quad (2)$$

Since we try out all possible subsets in step (5), we will get the desired result. We will argue in the following manner: consider the multi-set  $C' = \{\frac{16t}{\varepsilon} \text{ copies of each element in } C\}$ . We can interpret  $C'$  as a union of multi-sets  $C'_1, C'_2, \dots, C'_t$ , where  $C'_j = \{\frac{16}{\varepsilon} \text{ copies of each element in } C\}$ . Also, since  $M$  consists of  $\eta t$  independently sampled points, we can interpret  $M$  as a union of multi-sets  $M'_1, M'_2, \dots, M'_t$  where  $M'_1$  is the first  $\eta$  points sampled,  $M'_2$  is the second  $\eta$  points and so on. For all  $j = 1, \dots, t$ , let  $M_j = C'_j \cup (M'_j \cap X_j)$ .<sup>7</sup> We will show that for every  $j \in \{1, \dots, t\}$ , with probability at least  $(3/4)$ ,  $M_j$  contains a subset  $T_j$  of size  $\tau$  that satisfies eqn. (2). Note that  $T_j$ 's being disjoint follows from the definition of  $M_j$ . It will be sufficient to prove the following lemma.

► **Lemma 2.** Consider the sets  $M_1, \dots, M_t$  as defined above. For any  $j \in \{1, \dots, t\}$ ,

$$\Pr \left[ \exists T_j \subseteq M_j \text{ s.t. } |T_j| = \tau \text{ and } \left( \Phi(\mu(T_j), X_j) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(X_j) + \frac{\varepsilon}{2t} OPT \right) \right] \geq \frac{3}{4}.$$

---

<sup>7</sup>  $M'_j \cap X_j$  in this case, denotes those points in the multi-set  $M'_j$  that belongs to  $X_j$ .

The formal proof of the above lemma is deferred to the full version of the paper. The proof is through a case analysis that is based on whether  $\frac{\Phi(C, X_j)}{\Phi(C, X)}$  is large or small for a particular  $j \in \{1, \dots, t\}$ .

- *Case-I:*  $(\Phi(C, X_j) \leq \frac{\varepsilon}{6\alpha t} \cdot \Phi(C, X))$

The interpretation of this condition is that the points in  $X_j$  are close to centers in the center set  $C$ . This means that an appropriate convex combination of points in  $C$  will give a good center for  $X_j$ . More precisely, here we will show that there is a subset  $T_j \subseteq C'_j \subseteq M_j$  that satisfies eqn. (2).

- *Case-II:*  $(\Phi(C, X_j) > \frac{\varepsilon}{6\alpha t} \cdot \Phi(C, X))$

This is the case where all points in  $X_j$  do not have a close center in the center set  $C$ . If we can show that a  $D^2$ -sampled set with respect to center set  $C$  has a subset  $S$  that may be considered uniform sample from  $X_j$ , then we can use known results<sup>8</sup> to argue that  $M_j$  has a subset  $T_j$  such that  $\mu(T_j)$  is a good center for  $X_j$ . Note that since  $\frac{\Phi(C, X_j)}{\Phi(C, X)} > \frac{\varepsilon}{6\alpha t}$ , we can argue that if we  $D^2$ -sample  $\text{poly}(\frac{t}{\varepsilon})$  elements, then we will get a good representation from  $X_j$ . However, note that some of the points from  $X_j$  may have centers in  $C$  that are very close and hence will have a very small chance of being  $D^2$ -sampled. In such a case, no subset  $S$  of a  $D^2$ -sampled set will behave like a uniform sample from  $X_j$ . So, we need to argue more carefully taking into consideration the fact that there may be points in  $X_j$  for which the chance of being  $D^2$ -sampled is very small. Here is the high-level argument that we will make:

- Consider the set  $X'_j$  which is same as  $X_j$  except that points in  $X_j$  that are very close to  $C$  have been “collapsed” to their closest center in  $C$ .
- Argue that a good center for the set  $X'_j$  is a good center for  $X_j$ .
- Show that a convex combination of copies of centers in  $C$  (i.e.,  $C'_j$ ) and  $D^2$ -sampled points from  $X_j$  gives a good center for the set  $X'_j$ .

More precisely, in this case we will show that  $M_j$  contains a subset  $T_j$  such that  $\Phi(\mu(T_j), X_j) \leq (1 + \frac{\varepsilon}{2}) \cdot \Delta(X_j)$  and hence  $T_j$  also satisfies eqn. (2).

In order to discuss the applications of the **GoodCenters** algorithm, let us note some of its interesting properties. Note that the algorithm essentially runs in a single iteration. The outer loop of size  $2^t$  consists of independent iterations and can be executed independently. The rest of the algorithm clearly follows a single line of control and does not have dependencies. This allows us to design constant-pass streaming algorithms (using *reservoir sampling*) and parallel algorithms. The second useful property is that it finds a good list for *any* fixed set of  $t \leq k$  clusters (w.h.p.). This allows us to exploit the algorithm in certain contexts where once good centers for a few clusters have been chosen, choosing good centers for the remaining clusters becomes easier. We discuss the applications of our algorithm in the subsequent subsections.

An interesting point to note about the **GoodCenters** algorithm is that the  $k$ -center-set  $C$  that it takes as input is only a constant factor approximate solution for the classical  $k$ -means problem (i.e., unconstrained version) and not some constrained version. Note that we will use the algorithm for designing PTAS for various constrained versions but constant factor solutions for those are not required. So in some sense, the **GoodCenters** algorithm can be seen as an effective way of converting a constant factor approximate solution for the  $k$ -means problem to PTAS for various constrained versions. Let us now discuss the applications.

<sup>8</sup> We use a result from Inaba et al. [27] which says that the centroid of  $O(1/\varepsilon)$  uniformly sampled points from any dataset (w.h.p.) gives  $(1 + \varepsilon)$ -approximation with respect to the 1-means cost for the dataset.

## 1.1 Clustering under stability/separation

The worst-case complexity of the  $k$ -means problem is well understood. As discussed earlier, the problem is NP-hard and APX-hard. Hence, various *beyond worst-case* type results have been explored in the context of the  $k$ -means problem and one such direction is clustering under some “clusterability” condition. That is, design algorithms for datasets that satisfy some condition that captures the fact that the data is clusterable or in other words the data has some meaningful clusters. Clusterability is captured in various ways using notions such as *separability* and *stability*. Separability means that the target clusters are separated in some geometrical sense and stability means that the target clustering does not change under small perturbations of the input points. Separability and stability are closely related and in various contexts one implies the other. A lot of work has been done in the area of algorithm design for the  $k$ -means problem under various clusterability conditions. We will discuss these stability properties and their relationship in detail in the full version of the paper. It can be argued that the  $\beta$ -distributed property of Awasthi et al. [5] given below is one of the weaker separation properties. Hence, any result for datasets satisfying the  $\beta$ -distributed condition will have consequences for datasets satisfying stronger conditions. So the relevant question is: *Are there good algorithms for datasets under this condition?*

► **Definition 3** ( $\beta$ -distributed). *A  $k$ -means instance  $(X, k)$  is called  $\beta$ -distributed iff the following holds for any optimal clustering  $\{X_1^*, \dots, X_k^*\}$ :  $\forall i, \forall x \notin X_i^*, \|x - \mu(X_i^*)\|^2 \geq \beta \cdot \frac{OPT^*}{|X_i^*|}$ .*

Cohen-Addad and Schwiegelshohn [16] gave a local search based algorithm with neighbourhood size  $O(\beta^{-1} \cdot \text{poly}(\varepsilon^{-3}))$  that translates to an approximation with running time  $O\left(n^{\frac{1}{\beta} \text{poly}(\frac{1}{\varepsilon})}\right)$ .<sup>9</sup> Awasthi et al. [5] gave a PTAS for the  $k$ -means/median problems on datasets that satisfy the  $\beta$ -distributed assumption. The running time has polynomial dependence on the input parameters  $n, k, d$  and exponential dependence on  $\beta^{-1}$  and  $\varepsilon^{-1}$  ( $\varepsilon$  is the accuracy parameter). Even though they showed that the super-polynomial dependence on  $\beta^{-1}$  and  $\varepsilon^{-1}$  cannot be improved, improving the dependence on other input parameters was left as an open question. In this work, we address this by giving a faster PTAS for the  $k$ -means problem under the  $\beta$ -distributed notion. The running time of the algorithm for the  $k$ -means problem by Awasthi et al. [5] is  $O(dn^3)(k \log n)^{\text{poly}(\frac{1}{\beta}, \frac{1}{\varepsilon})}$ . We improve the running time to  $O\left(dn^3 \left(\frac{k}{\varepsilon}\right)^{O(\frac{1}{\beta\varepsilon^2})}\right)$ . Note that due to our improvement in running time, our algorithm is also a Fixed Parameter Tractable Approximation Scheme (FPT-AS) for the problem with parameters  $k$  and  $\beta$ . Moreover, the running time does not have an exponential dependence on  $k$  that is typically the case for such FPT approximation schemes for general datasets. We formally state our result as the following theorem. We shall discuss the proof of this theorem in the full version of the paper.

► **Theorem 4.** *Let  $\varepsilon, \beta > 0$ ,  $k$  be a positive integer, and let  $X \subset \mathbb{R}^d$  be a  $\beta$ -distributed dataset. There is an algorithm that takes as input  $(X, k, \varepsilon, \beta)$  and outputs a  $k$ -center-set  $C$  such that  $\Phi(C, X) \leq (1 + \varepsilon) \cdot OPT^*$  and the algorithm runs in time  $O\left(dn^3 \left(\frac{k}{\varepsilon}\right)^{O(\frac{1}{\beta\varepsilon^2})}\right)$ .*

Our running time improvements over the algorithm of Awasthi et al. [5] comes from using a faster algorithm to find good centers for a few (constant) optimal clusters called “expensive clusters” in the terminology used by Awasthi et al. They had pointed out that if there was

<sup>9</sup> It may be tempting to think that using the local search algorithm on a coresets (instead of the dataset) will improve the running time to  $O(nkd + k^{\frac{1}{\beta} \text{poly}(\frac{1}{\varepsilon})})$ . However, it is important to realise that known coresets constructions that give coresets of size  $\text{poly}(k, 1/\varepsilon)$  may not be stability/separation preserving.

a faster algorithm for finding good centers for these expensive clusters, then the overall running time of their algorithm could be significantly improved. This is precisely what our `GoodCenters` algorithm allows us to do. The `GoodCenters` algorithm creates a list such that at least one element in the list is a set of good centers for the expensive clusters. So, one can execute the algorithm of Awathi et al. repeatedly for every element of the list and then pick the best solution. The details are given in the full version of the paper.

## 1.2 Parameterised reduction from outlier $k$ -means to $k$ -means

The  $k$ -means problem models the clustering problem when the data is *noise-free*. That is, the data does not contain outlier points. Clustering algorithms designed for noise-free datasets may behave badly when used for datasets with outliers, where the objective is to cluster the non-outlier points. This is because clustering objective functions such as  $k$ -means/median may be sensitive to outliers. This motivates modelling noisy data clustering as a separate problem. One way to model noisy data clustering is through a problem known as *outlier  $k$ -means* or  $k$ -means with outliers problem. This problem has been studied in a number of previous works [11, 13, 12, 31, 22, 8, 25]. The problem is formally defined as:

**Outlier  $k$ -means:** Given a set of  $n$  points  $X \subset \mathbb{R}^d$  and positive integers  $k, m$ , find a set of  $k$  centers  $C \subset \mathbb{R}^d$  such that the following cost function is minimised:  

$$\Phi_o(C, X) \equiv \min_{Z \subseteq X, |Z|=m} \left( \sum_{x \in X \setminus Z} \min_{c \in C} \|x - c\|^2 \right).$$

This is the same as optimising the  $k$ -means cost function on all but at most  $m$  points which can be interpreted as outliers. Note that once an optimal center-set  $C$  is obtained, the outliers can be located as the farthest  $m$  points from the centers in  $C$ . In the other direction, suppose we know the  $m$  outlier points  $Z \subseteq X$ , then the optimal center set  $C$  may be found by solving the  $k$ -means problem on the dataset  $X \setminus Z$ . The classical  $k$ -means problem can be considered a special case of this general problem where  $m = 0$ . So, the known hardness results for  $k$ -means naturally holds for outlier  $k$ -means as well. Given this, an interesting problem is to analyse the relative hardness of these problems. In other words, is the outlier  $k$ -means problem harder than the classical  $k$ -means problem in some sense? One way to formalise this question is to ask whether the outlier version becomes easier if there is an oracle for the  $k$ -means problem? In other words, is there an efficient reduction from the outlier- $k$ -means problem to the  $k$ -means problem? One brute-force reduction is to consider all possible subsets of  $m$  outliers and then solve the  $k$ -means problem on the remaining points. However, the running time of this reduction is  $\binom{n}{m} = O(n^m)$  which is prohibitively large.

The same question regarding the relative hardness of these problem can also be asked in the approximation setting. The known results on efficient approximation algorithms for these problems makes this question interesting even in the approximation setting. There is a gap in approximation guarantee between the best known efficient approximation algorithm for  $k$ -means and outlier  $k$ -means. The best known polynomial time approximation guarantee for the  $k$ -means problem is 6.358 [2] and for  $k$ -means with outliers is 53.003 [31]. So the relevant question is whether this gap can be removed. We initiate the discussion by giving a parameterised reduction from the outlier  $k$ -means problem to the  $k$ -means problem. We give a parameterised reduction from the approximate  $k$ -means with outliers problems with parameters  $k, m$ , and  $\frac{1}{\varepsilon}$  to the classical  $k$ -means problem.

► **Theorem 5.** *Let  $0 < \varepsilon \leq \frac{1}{2}$ . Let  $\mathcal{M}$  be an oracle that returns an optimal solution for arbitrary instances of the  $k$ -means problem. Then there exists an algorithm `OutlierAlg` <sup>$\mathcal{M}$</sup>  $(X, k, m, \varepsilon)$  that returns a  $(1 + \varepsilon)$ -approximate solution to the outlier  $k$ -means problem with probability*

## 13:10 On Sampling Based Algorithms for $k$ -Means

at least  $\frac{3}{4}$ , where  $X \subset \mathbb{R}^d$  and  $k, m$  are positive integers. The number of calls made to the oracle  $\mathcal{M}$  is bounded by  $|\mathcal{L}| = \left(\frac{k+m}{\varepsilon}\right)^{O\left(\frac{m}{\varepsilon^2}\right)}$  and the running time of the algorithm is bounded by  $O(nd \cdot |\mathcal{L}|)$ .

The main idea is to consider the  $m$  outliers in an optimal solution as clusters of their own. We can then treat  $k$  optimal clusters along with these  $m$  outlier clusters as the partitioning  $X_1, \dots, X_{k+m}$  of the dataset. The **GoodCenters** algorithm, when executed with  $t = m$ , is guaranteed (w.h.p.) to output a list of  $m$ -center-sets such that at least one is good for the outlier-clusters. This means that at least one of the  $m$ -center-sets will be such that the  $m$  centers are close to the outliers. We can exploit this fact to locate good outliers for the dataset, remove them, and solve the  $k$ -means problem on the remaining instance. However, since we will need to try all  $m$ -center-sets in the list produced by the **GoodCenters** algorithm, we will pay in terms of the running time with a multiplicative factor proportional to the list size. Replacing the  $k$ -means oracle  $\mathcal{M}$  with a more realistic constant  $c$ -approximation algorithm  $\mathcal{A}$  for  $k$ -means with running time  $t(n, k, d)$ , we obtain a  $(c + c\varepsilon)$ -approximation algorithm **OutlierAlg<sup>A</sup>** for outlier  $k$ -means with running time  $O\left(t(n, k, d) \cdot \left(\frac{k+m}{\varepsilon}\right)^{O\left(\frac{m}{\varepsilon^2}\right)}\right)$ . The consequences of this is that it removes the approximation factor gap between the  $k$ -means and outlier  $k$ -means problem at the cost of increasing the running time by a factor of  $\left(\frac{k+m}{\varepsilon}\right)^{O\left(\frac{m}{\varepsilon^2}\right)}$ . However, one should note that this factor is independent of the problem size and is small when compared to the brute-force reduction (considering all possible subsets of  $m$  outliers) with associated factor of  $O(n^m)$ .

Using the **GoodCenters** algorithm in a different manner in the outlier setting gives us another interesting consequence. The **GoodCenters** algorithm, when executed with  $t = k$  is guaranteed (w.h.p.) to output a list of  $k$ -center-sets such that at least one is good for the  $k$  optimal clusters. This gives an FPT-approximation scheme (with parameters  $k$  and  $m$ ) for the outlier  $k$ -means problem with running time  $O(nd \cdot f(k, m, \varepsilon))$  and furthermore a 4-pass streaming algorithm that uses  $O(f(k, m, \varepsilon) \cdot \log n)$ -space, where  $f(k, m, \varepsilon) = O\left(nd \left(\frac{m+k}{\varepsilon}\right)^{O\left(\frac{k}{\varepsilon}\right)}\right)$ . The details of this section are given in the full version of the paper.

### 1.3 Streaming algorithms for constrained versions of $k$ -means

We discussed how an algorithm for the list- $k$ -means problem can be converted to a PTAS for a constrained  $k$ -means problem given that there is a *partition algorithm* that finds a feasible clustering with the smallest  $k$ -means cost. Examining the **GoodCenters** algorithm closely, we realise that it can be implemented in 2-passes using small amount of space. This opens the door for designing streaming PTAS for the constrained versions of the  $k$ -means problem. If one can design a streaming version of the partition algorithm for some constrained  $k$ -means problem, then combining it with the streaming version of the **GoodCenters** algorithm will give us a streaming PTAS for the problem. So, let us first discuss how a streaming version of the **GoodCenters** algorithm can be designed.

The first bottleneck in designing a streaming version of **GoodCenters** is that we need a constant factor approximate solution  $C$  for the  $k$ -means problem (i.e., the unconstrained  $k$ -means problem). Fortunately, there exists a 1-pass, logspace streaming algorithm that gives a constant factor approximate solution to the  $k$ -means problem [10]. Given  $C$ , we need to show how to implement step (3) of the algorithm in a streaming manner (the  $2^t$  repetitions can be performed independently, this appears as a multiplicative factor in the space used). The probability of sampling a point  $p$  is proportional to  $\Phi(C, p)$ , with the constant of proportionality being  $\Phi(C, X)$ . The sampling can be performed using the ideas

of *reservoir sampling* (see e.g. [40]). Since we need to sample  $\eta t \leq \text{poly}(\frac{k}{\varepsilon})$  points in step (3), reservoir sampling takes  $O(\text{poly}(\frac{k}{\varepsilon}) \cdot \log n)$  space. Given a sample  $M$ , steps (5)-(6) can be implemented in  $O(|M|^{k\tau})$  space, where  $\tau = O(\frac{1}{\varepsilon})$ . This can be summarised formally as the following useful lemma that we will prove in the full version of the paper (we assume that storing a point accounts for one unit of space).

► **Lemma 6.** *The algorithm `GoodCenters` can be implemented using 2-passes over the input data while maintaining space of  $O(f(k, \varepsilon) \cdot \log n)$ , where  $f(k, \varepsilon) = (\frac{k}{\varepsilon})^{O(\frac{k}{\varepsilon})}$ .*

Let us now see how to design a streaming PTAS for a constrained  $k$ -means problem using the above lemma. Let  $\mathcal{P}^C$  denote the partition algorithm for this constrained problem and suppose there is a streaming version  $\mathcal{SP}^C$  of this partition algorithm. We will use the 2-pass streaming version of the `GoodCenters` algorithm to output the list  $\mathcal{L}$ . We will then use  $\mathcal{SP}^C$  on each element of  $\mathcal{L}$  (independently) and pick the best solution. Since  $|\mathcal{L}|$  is small, so is the space requirement. From the previous discussion, we know that (w.h.p.) we are guaranteed to obtain a  $(1 + \varepsilon)$ -approximate solution. Hence we get a constant pass streaming PTAS. So, as long as there is a streaming partition algorithm for a constrained  $k$ -means problem, there is also a streaming PTAS. Now the question is whether there are constrained  $k$ -means problems for which such streaming partition algorithms can be designed. Interestingly, we can design such streaming partition algorithms for four out of the six constrained  $k$ -means problems in Table 1. Our results can be summarised as the following main theorem the proof of which is deferred to the full version of the paper. Here,  $\Delta$  is the *aspect ratio*, i.e.,  $\Delta = \frac{\max_{p \in X, c \in C} \|p - c\|}{\min_{p \in X \setminus C, c \in C} \|p - c\|}$ .

► **Theorem 7.** *There is a  $(1 + \varepsilon)$ -approximate, 4-pass, streaming algorithm for the following constrained  $k$ -means clustering problems that uses  $O(f(k, \varepsilon) \cdot (\log \Delta + \log n))$ -space and  $O(d \cdot f(k, \varepsilon))$  time per item, where  $f(k, \varepsilon) = (\frac{k}{\varepsilon})^{O(\frac{k}{\varepsilon})}$ :*

1.  $k$ -means
2.  $r$ -gather  $k$ -means
3.  $r$ -capacity  $k$ -means
4. Fault tolerant  $k$ -means
5. Semi-supervised  $k$ -means

Further, the space requirement can be improved to  $O(f(k, \varepsilon) \cdot \log n)$  using 5-passes.

Note that the classical  $k$ -means problem can also be seen as a constrained  $k$ -means problem where there are no constraints. Also note that two constrained versions of constrained  $k$ -means problems from Table 1 are missing from the theorem above. These are the *chromatic  $k$ -means clustering* and the  *$l$ -diversity clustering*. We can show that deterministic logspace streaming algorithms for these problems are not possible. Due to space limitations, this is shown in the full version of the paper.

### Comparison with Coreset based streaming algorithms

Streaming *coreset* constructions provide another approach to designing streaming algorithm for the  $k$ -means problem. An  $(\varepsilon, k)$  coreset of a dataset  $X \subset \mathbb{R}^d$  is a weighted set  $S \subset \mathbb{R}^d$  along with a weight function  $w : S \rightarrow \mathbb{R}^+$  such that for any  $k$ -center-set  $C$ , we have:  $|\sum_{s \in S} \min_{c \in C} w(s) \cdot \|s - c\|^2 - \sum_{x \in X} \min_{c \in C} \|x - c\|^2| \leq \varepsilon \cdot \sum_{x \in X} \min_{c \in C} \|x - c\|^2$ . So, it is sufficient to find good  $k$ -center-set for a coreset  $S$  (instead of the dataset  $X$ ). There exists one-pass streaming coreset construction [19] that uses  $\text{poly}(k, \frac{1}{\varepsilon}, \log n)$  space and outputs a coreset of size  $\text{poly}(k, \frac{1}{\varepsilon}, \log n)$ . Using this, one can design a single-pass streaming algorithm for the  $k$ -means problem by first running the streaming algorithm to output a coreset and then finding a good  $k$  center set for the small coreset. If the output is supposed to be a clustering, then we will need to make another pass over the data. Note that the same idea of

working on coresets does not trivially carry over to the constrained versions of  $k$ -means as there are additional constraints. However, there is a specific geometric coreset construction which works for constrained versions of  $k$ -means. This is one of the first coreset constructions for  $k$ -means by Har-Peled and Mazumdar [26] where the points in the coreset are such that the sum total of the distance of the data points to the nearest coreset point is small. The weight of a coreset point is simply the number of data points for which the coreset point is the closest. So, a coreset point *represents* a subset of data points. Schmidt et al. [38] used this construction for a constrained version called Fair  $k$ -means. This coreset construction can be performed in a single pass over the data. The coreset size is  $O(k\varepsilon^{-d} \log n)$  and it can be computed in as much space using ideas developed later (e.g., [20]). Even though this gives a one-pass algorithm for producing a good center set (two passes for producing clustering), the space requirement is exponentially large in the dimension. Fortunately, in a more recent development by Makarychev et al. [37] showed that the  $k$ -means cost of *any* clustering is preserved up to a factor of  $(1 + \varepsilon)$  under a projection onto a random  $O\left(\frac{\log(k/\varepsilon)}{\varepsilon^2}\right)$ -dimensional subspace. This result when combined with the geometric coreset construction of Har-Peled and Mazumdar [26] gives a one-pass,  $O\left(\left(\frac{k}{\varepsilon}\right)^{\frac{1}{\varepsilon^2}} \cdot \log n\right)$ -space algorithm for producing a good  $k$ -center-set for *any* constrained version of the  $k$ -means problem. Even though the space bound has a slightly worse dependency on  $1/\varepsilon$  than our list- $k$ -means based idea, the dependency on  $k$  and number of passes is much better. Indeed, we overlooked this connection with coreset of Har-Peled and Mazumdar and dimension reduction of Makarychev et al. when we were designing our list- $k$ -means based streaming algorithms and were made to realise this at a later stage of this work. At this point, all we can say is that designing streaming algorithm based on list- $k$ -means is another way of approaching constrained  $k$ -means problem. Furthermore, we decided to include this section since some of the techniques developed here may have independent applications. We also note that coreset based technique does not seem to work for the constrained binary  $k$ -means which is also a problem does not fit into the unified framework of Ding and Xu [18]. This is because current known techniques for finding good centers for this problem requires uniform samples from the optimal clusters and it is not clear whether working with representative points (as in the coreset) will work. We discuss constrained binary  $k$ -means and a related problem next.

#### 1.4 Streaming algorithms for binary- $k$ -means and low rank approximation

Low rank approximation is a common data analysis task. The most general version of the problem, the  $\ell_p$ -low rank approximation problem, is defined in the following manner:

$\ell_p$ -low rank approximation: Given a matrix  $\mathbf{A} \in \mathbb{R}^{n \times d}$  (with  $n \geq d$ ) and an integer  $r$ , find a rank- $r$  matrix  $\mathbf{B} \in \mathbb{R}^{n \times d}$  such that  $\|\mathbf{A} - \mathbf{B}\|_p^p \equiv \sum_{i,j} |\mathbf{A}_{i,j} - \mathbf{B}_{i,j}|^p$  is minimised.

The above definition is for any positive value of  $p$ . When  $p = 0$ , the objective is to minimise  $\|\mathbf{A} - \mathbf{B}\|_0$  which is defined to be the number of mis-matches in the matrices  $\mathbf{A}$  and  $\mathbf{B}$ . The  $\ell_p$ -low rank approximation problem is known to be NP-hard for  $p \in \{0, 1\}$  while for  $p = 2$  the problem can be solved using SVD (Singular Value Decomposition). The specific case of  $p = 0$  is known as the  $\ell_0$ -low rank approximation problem. The problem can alternatively be stated as: given an  $n \times d$  matrix  $\mathbf{A}$ , find an  $n \times r$  matrix  $\mathbf{U}$  and a  $r \times d$  matrix  $\mathbf{V}$  such that  $\|\mathbf{A} - \mathbf{U} \cdot \mathbf{V}\|_0$  is minimised. There is an interest in specific class of instances of the  $\ell_0$ -low rank approximation problem where the matrices  $\mathbf{A}$ ,  $\mathbf{U}$ ,  $\mathbf{V}$  are binary matrices. In fact, we can generalise even further by making the notion of  $\mathbf{U} \cdot \mathbf{V}$  in the above definition more flexible in

the following manner: If  $\mathbf{A}' = \mathbf{U} \cdot \mathbf{V}$ , then  $\mathbf{A}'_{ij}$  is the inner product of the  $i^{\text{th}}$  row of  $\mathbf{U}$  and the  $j^{\text{th}}$  column of  $\mathbf{V}$ . We can consider various fields for this inner product. The two popularly explored fields are: (i)  $\mathbb{F}_2$  with inner product defined as  $\langle x, y \rangle \equiv \oplus_i (x_i \cdot y_i)$ , and (ii) Boolean semiring  $\{0, 1, \wedge, \vee\}$  with inner product defined as  $\langle x, y \rangle \equiv \vee_i (x_i \wedge y_i) = 1 - \prod_i (1 - x_i \cdot y_i)$ . We can generalise the problem (using the formulation in terms of  $\mathbf{U}$  and  $\mathbf{V}$ ) so that the above versions become special cases. This was done by Ban et al. [7] and they called this problem *generalised binary  $\ell_0$ -rank- $r$  problem* that is defined below.

*Generalised binary  $\ell_0$ -rank- $r$  approximation:* Given a matrix  $\mathbf{A} \in \{0, 1\}^{n \times d}$  with  $n \geq d$ , an integer  $r$ , and an inner product function  $\langle \cdot, \cdot \rangle : \{0, 1\}^r \times \{0, 1\}^d \rightarrow \{0, 1\}$ , find matrices  $\mathbf{U} \in \{0, 1\}^{n \times r}$  and  $\mathbf{V} \in \{0, 1\}^{r \times d}$  that minimises  $\|\mathbf{A} - \mathbf{U} \cdot \mathbf{V}\|_0$ , where  $\mathbf{U} \cdot \mathbf{V}$  is computed using the inner product function. That is  $[\mathbf{U} \cdot \mathbf{V}]_{ij}$  is the inner product of the  $i^{\text{th}}$  row of  $\mathbf{U}$  with the  $j^{\text{th}}$  column of  $\mathbf{V}$ .

Ban et al. [7] showed that there is no approximation algorithm for the generalised binary  $\ell_0$ -rank- $r$  problem running in time  $2^{2^{\delta r}}$  for a constant  $\delta > 0$  even though faster algorithms are known for certain specific versions [33]. The work of Ban et al. [7] and Fomin et al. [21] addressed one of the main open questions for generalised binary  $\ell_0$  rank- $r$  problem – whether a PTAS for constant  $r$  is possible. They give such a PTAS using very similar set of ideas (even though they were obtained independently). We extend the previous work of Ban et al. and Fomin et al. to the streaming setting by using the connection of this problem to the *constrained binary  $k$ -means problem* which we discuss next. This connection was given and used by both Ban et al. [7] and by Fomin et al. [21]. We will work with the definition of the constrained binary  $k$ -means problem given by Fomin et al. [21]. For this, we first need to define the concept of a set of  $k$  centers  $C \subseteq \{0, 1\}^d$  satisfying a set of  $k$ -ary relations. Given a set  $\mathcal{R} = \{R_1, \dots, R_d\}$  of  $d$ ,  $k$ -ary binary relations (i.e.,  $R_i \subseteq \{0, 1\}^k$  for every  $i$ ), a set  $C = \{c_1, \dots, c_k\} \subseteq \{0, 1\}^d$  of  $k$  centers is said to satisfy  $\mathcal{R}$  iff  $(c_1[i], \dots, c_k[i]) \in R_i$  for every  $i = 1, \dots, d$ . Here,  $c_j \in \{0, 1\}^d$  is thought of as a  $d$ -dimensional vector and  $c_j[i]$  denotes the  $i^{\text{th}}$  coordinate of this vector. We can now define the constrained binary  $k$ -means problem.

*Constrained binary  $k$ -means:* Given a set of  $n$  points  $X \subseteq \{0, 1\}^d$ , a positive integer  $k$ , and a set of  $k$ -ary relations  $\mathcal{R} = \{R_1, \dots, R_d\}$ , find a set of  $k$  centers  $C \subseteq \{0, 1\}^d$  satisfying  $\mathcal{R}$  such that the cost function  $\Phi(C, X) \equiv \sum_{x \in X} \min_{c \in C} \|x - c\|_2^2 = \sum_{x \in X} \min_{c \in C} \mathcal{H}(x, c)$  is minimised. Here  $\mathcal{H}(\cdot, \cdot)$  denotes the Hamming distance.

It is important to distinguish between the definition of constrained binary  $k$ -means problem given above with the constrained  $k$ -means problem discussed earlier. The relevant question to ask is: *Does the constrained binary  $k$ -means problem fit into the unified framework of Ding and Xu [18]?* If the answer to the above question were yes, then a streaming PTAS for the constrained binary  $k$ -means problem would trivially follow from the earlier discussion on constrained  $k$ -means. Unfortunately, this is not true. Note that the framework of Ding and Xu [18] defines the constraints on the clusters while the definition of constrained binary  $k$ -means problem defines constraints on the centers. However, we note that the  $D^2$ -sampling based techniques can be extended to this setting. Below, we formally state our main results for the constrained binary- $k$ -means problem.

► **Theorem 8.** *Let  $0 < \varepsilon \leq 1/2$ . There is a 3-pass streaming algorithm that outputs a  $(1 + \varepsilon)$ -approximate solution for any instance of the constrained binary  $k$ -means problem. The space and per-item processing time of our algorithm is  $O\left(d \cdot (\log n)^k \cdot 2^{\tilde{O}\left(\frac{k^2}{\varepsilon^2}\right)}\right)$ .*

Note that as per the formulation of the constrained binary  $k$ -means problem, the output is supposed to be a set of  $k$  centers. The above 3-pass algorithm outputs such a  $k$ -center-set. However, if the objective is to output the clustering of the data points  $X$ , then one more pass over the data will be required and the resulting algorithm will be a 4-pass algorithm. This is relevant for the generalised binary  $\ell_0$ -rank- $r$  approximation problem that we discuss next. We obtain a result for the generalised binary  $\ell_0$ -rank- $r$  problem that is similar to the above result, using a simple reduction. This reduction is used by both Fomin et al. [21] and Ban et al. [7]. We restate the result of Fomin et al. [21] for clarity.

► **Lemma 9** (Lemma 1 and 2 of [21]). *For any instance  $(\mathbf{A}, r)$  of the generalised binary  $\ell_0$ -rank- $r$  approximation problem, one can construct in time  $O(n + d + 2^{2r})$  an instance  $(X, k = 2^r, \mathcal{R})$  of constrained binary  $k$ -means problem with the following property: Given any  $\alpha$ -approximate solution  $C$  of  $(X, k, \mathcal{R})$ , an  $\alpha$ -approximate solution  $\mathbf{B}$  of  $(\mathbf{A}, r)$  can be constructed in time  $O(rnd)$ .*

The dataset  $X$  corresponding to matrix  $\mathbf{A}$ , in the above reduction, is essentially the rows of the matrix  $\mathbf{A}$  and  $\forall i, R_i = \{(\langle x, \lambda_1 \rangle, \dots, \langle x, \lambda_k \rangle) : x \in \{0, 1\}^r\}$  and  $\lambda_i$ 's are pairwise distinct vectors in  $\{0, 1\}^r$ . The above reduction and Theorem 8 gives the following main result for the generalised binary  $\ell_0$ -rank- $r$  approximation problem. Note that since we need to output a matrix  $\mathbf{B}$ , we will need the clustering of the rows of  $\mathbf{A}$  and as per previous discussion this will require one more pass than that in Theorem 8.

► **Theorem 10.** *Let  $0 < \varepsilon \leq 1/2$ . There is a 4-pass streaming algorithm that makes row-wise passes over the input matrix and outputs a  $(1 + \varepsilon)$ -approximate solution for any instance of the generalised binary  $\ell_0$ -rank- $r$  problem. The space and per-item processing time of our algorithm is  $O\left(d \cdot (\log n)^{2^r} \cdot 2^{\tilde{O}\left(\frac{2^{2r}}{\varepsilon^2}\right)}\right)$ .*

The details of this section are given in the full version of the paper.

## 1.5 Conclusion and open problems

Our results demonstrate the versatility of the sampling based approach for  $k$ -means. This has also been demonstrated in some of the past works. The effectiveness of  $k$ -means++ (which is basically  $D^2$ -sampling in  $k$  rounds) is well known [4]. The  $D^2$ -sampling technique has been used to give simple PTAS for versions of the  $k$ -means/median problems with various metric-like distance measures [28] and also various constrained variations of  $k$ -means [9]. It has also been used to give efficient algorithms in the semi-supervised setting [3, 24] and coresets construction [35]. In this work, we see its use in the streaming, outlier, and clustering-under-stability settings. The nice property of the sampling based approach is that we have a uniform template of the algorithm that is simple and that works in various different settings. This essentially means that the algorithm remains the same while the analysis changes.

This work raises many interesting questions. Our main result on list- $k$ -means is a sampling algorithm that helps us find good centers for *any* subset of  $t$  clusters. We made use of this property in clustering-under-stability and outlier settings. There may be other such settings where the clustering problem may become easier once good centers for a few clusters have been chosen. Our discussion on outlier  $k$ -means raises an interesting question related to the relative hardness of the  $k$ -means and the outlier  $k$ -means problem. In the streaming setting for the constrained  $k$ -means, we give a generic algorithm within the unified framework of Ding and Xu [18]. The advantage of working in this unified framework is that we get streaming algorithms for various constrained versions of the  $k$ -means problem. However, it

may be possible to obtain better streaming algorithms (in terms of space, time, and number of passes) for the constrained problems when considered separately as is the case for the classical  $k$ -means problem [10]. It may be worthwhile exploring these problems.

---

## References

- 1 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for  $k$ -means clustering. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX '09 / RANDOM '09*, pages 15–28, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-03685-9\_2.
- 2 S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for  $k$ -means and euclidean  $k$ -median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, October 2017. doi:10.1109/FOCS.2017.15.
- 3 Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Approximate Clustering with Same-Cluster Queries. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:21, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.40.
- 4 David Arthur and Sergei Vassilvitskii.  $k$ -means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '07*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283494>.
- 5 Pranjal Awasthi, Avrim Blum, and Or Sheffet. Stability yields a PTAS for  $k$ -median and  $k$ -means clustering. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS '10*, pages 309–318, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.36.
- 6 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The Hardness of Approximation of Euclidean  $k$ -Means. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 754–767, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SOCG.2015.754.
- 7 Frank Ban, Vijay Bhattiprolu, Karl Bringmann, Pavel Kolev, Euiwoong Lee, and David P. Woodruff. A PTAS for  $\ell_p$ -low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '19*, pages 747–766, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3310435.3310482>.
- 8 Aditya Bhaskara, Sharvaree Vadgama, and Hong Xu. Greedy sampling for approximate clustering in the presence of outliers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11146–11155. Curran Associates, Inc., 2019. URL: <http://papers.nips.cc/paper/9294-greedy-sampling-for-approximate-clustering-in-the-presence-of-outliers.pdf>.
- 9 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster algorithms for the constrained  $k$ -means problem. *Theory of Computing Systems*, 62(1):93–115, January 2018.
- 10 Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming  $k$ -means on well-clusterable data. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '11*, pages 26–40, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=2133036.2133039>.

## 13:16 On Sampling Based Algorithms for $k$ -Means

- 11 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 642–651, USA, 2001. Society for Industrial and Applied Mathematics.
- 12 Sanjay Chawla and Aristides Gionis. *k-means: A unified approach to clustering and outlier detection*, pages 189–197. Society for Industrial and Applied Mathematics, 2013. doi:10.1137/1.9781611972832.21.
- 13 Ke Chen. A constant factor approximation algorithm for  $k$ -median clustering with outliers. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 826–835, USA, 2008. Society for Industrial and Applied Mathematics.
- 14 V. Cohen-Addad and Karthik C.S. Inapproximability of clustering in  $l_p$  metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 519–539, November 2019. doi:10.1109/FOCS.2019.00040.
- 15 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for  $k$ -means and  $k$ -median in euclidean and minor-free metrics. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, 00:353–364, 2016. doi:10.1109/FOCS.2016.46.
- 16 Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 49–60, 2017. doi:10.1109/FOCS.2017.14.
- 17 Sanjoy Dasgupta. The hardness of  $k$ -means clustering. Technical Report CS2008-0916, Department of Computer Science and Engineering, University of California San Diego, 2008.
- 18 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1471–1490, 2015. doi:10.1137/1.9781611973730.97.
- 19 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for  $k$ -means clustering based on weak coresets. In *Proceedings of the twenty-third annual symposium on Computational geometry*, SCG '07, pages 11–18, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247072.
- 20 Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Bico: Birch meets coresets for  $k$ -means clustering. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 481–492, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- 21 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Approximation schemes for low-rank binary matrix approximation problems. *CoRR*, abs/1807.07156, 2018. arXiv:1807.07156.
- 22 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. *ACM Trans. Algorithms*, 15(2), February 2019. doi:10.1145/3301446.
- 23 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for  $k$ -means in doubling metrics. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, 00:365–374, 2016. doi:10.1109/FOCS.2016.47.
- 24 Buddhima Gamlath, Sangxia Huang, and Ola Svensson. Semi-Supervised Algorithms for Approximately Optimal and Accurate Clustering. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.57.
- 25 Shalmoli Gupta, Ravi Kumar, Kefu Lu, Benjamin Moseley, and Sergei Vassilvitskii. Local search methods for  $k$ -means with outliers. *Proc. VLDB Endow.*, 10(7):757–768, March 2017. doi:10.14778/3067421.3067425.

- 26 Sariel Har-Peled and Soham Mazumdar. On coresets for  $k$ -means and  $k$ -median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 291–300, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007400.
- 27 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based  $k$ -clustering: (extended abstract). In *Proceedings of the tenth annual symposium on Computational geometry*, SCG '94, pages 332–339, New York, NY, USA, 1994. ACM. doi:10.1145/177424.178042.
- 28 Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple  $D^2$ -sampling based PTAS for  $k$ -means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014. doi:10.1007/s00453-013-9833-9.
- 29 Ragesh Jaiswal, Mehul Kumar, and Pulkit Yadav. Improved analysis of  $D^2$ -sampling based PTAS for  $k$ -means and other clustering problems. *Information Processing Letters*, 115(2):100–103, 2015. doi:10.1016/j.ip1.2014.07.009.
- 30 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for  $k$ -means clustering. In *Proc. 18th Annual Symposium on Computational Geometry*, pages 10–18, 2002. doi:10.1145/513400.513402.
- 31 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for  $k$ -median and  $k$ -means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 646–659, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188882.
- 32 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, February 2010. doi:10.1145/1667053.1667054.
- 33 Ravi Kumar, Rina Panigrahy, Ali Rahimi, and David Woodruff. Faster algorithms for binary matrix factorization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3551–3559, Long Beach, California, USA, 09–15 June 2019. PMLR. URL: <http://proceedings.mlr.press/v97/kumar19a.html>.
- 34 Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, June 1995. doi:10.1007/BF01200757.
- 35 Mario Lucic, Matthew Faulkner, Andreas Krause, and Dan Feldman. Training gaussian mixture models at scale via coresets. *J. Mach. Learn. Res.*, 18(1):5885–5909, January 2017. URL: <http://dl.acm.org/citation.cfm?id=3122009.3242017>.
- 36 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar  $k$ -means problem is NP-hard. *Theor. Comput. Sci.*, 442:13–21, July 2012. doi:10.1016/j.tcs.2010.05.034.
- 37 Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for  $k$ -means and  $k$ -medians clustering. *CoRR*, abs/1811.03195, 2018. arXiv:1811.03195.
- 38 Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair  $k$ -means. In Evripidis Bampis and Nicole Megow, editors, *Approximation and Online Algorithms*, pages 232–251, Cham, 2020. Springer International Publishing.
- 39 Andrea Vattani. The hardness of  $k$ -means clustering in the plane. Technical report, Department of Computer Science and Engineering, University of California San Diego, 2009.
- 40 J S Vitter. Random sampling with a reservoir. *ACM Trans. Math. Software*, 11(1):37–57, 1985.



# String Indexing for Top- $k$ Close Consecutive Occurrences

**Philip Bille** 

Technical University of Denmark, DTU Compute, Lyngby, Denmark  
phbi@dtu.dk

**Inge Li Gørtz** 

Technical University of Denmark, DTU Compute, Lyngby, Denmark  
inge@dtu.dk

**Max Rishøj Pedersen** 

Technical University of Denmark, DTU Compute, Lyngby, Denmark  
mhrpe@dtu.dk

**Eva Rotenberg** 

Technical University of Denmark, DTU Compute, Lyngby, Denmark  
erot@dtu.dk

**Teresa Anna Steiner** 

Technical University of Denmark, DTU Compute, Lyngby, Denmark  
terst@dtu.dk

---

## Abstract

The classic string indexing problem is to preprocess a string  $S$  into a compact data structure that supports efficient subsequent pattern matching queries, that is, given a pattern string  $P$ , report all occurrences of  $P$  within  $S$ . In this paper, we study a basic and natural extension of string indexing called the string indexing for top- $k$  close consecutive occurrences problem (SITCCO). Here, a consecutive occurrence is a pair  $(i, j)$ ,  $i < j$ , such that  $P$  occurs at positions  $i$  and  $j$  in  $S$  and there is no occurrence of  $P$  between  $i$  and  $j$ , and their distance is defined as  $j - i$ . Given a pattern  $P$  and a parameter  $k$ , the goal is to report the top- $k$  consecutive occurrences of  $P$  in  $S$  of minimal distance. The challenge is to compactly represent  $S$  while supporting queries in time close to the length of  $P$  and  $k$ . We give two time-space trade-offs for the problem. Let  $n$  be the length of  $S$ ,  $m$  the length of  $P$ , and  $\epsilon \in (0, 1]$ . Our first result achieves  $O(n \log n)$  space and optimal query time of  $O(m + k)$ , and our second result achieves linear space and query time  $O(m + k^{1+\epsilon})$ . Along the way, we develop several techniques of independent interest, including a new translation of the problem into a line segment intersection problem and a new recursive clustering technique for trees.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pattern matching; Theory of computation  $\rightarrow$  Data structures design and analysis

**Keywords and phrases** String indexing, pattern matching, consecutive occurrences

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.14

**Related Version** <https://arxiv.org/abs/2007.04128>.

**Funding** *Philip Bille*: Supported by the Danish Research Council grant DFF-8021-002498.

*Inge Li Gørtz*: Supported by the Danish Research Council grant DFF-8021-002498.

*Max Rishøj Pedersen*: Supported by the Danish Research Council grant DFF-8021-002498.

*Eva Rotenberg*: Supported by the Danish Research Council grant DFF-8021-002498.

**Acknowledgements** We thank anonymous reviewers of an earlier draft of this paper for their insightful comments and suggestions for improvement.



© Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, Eva Rotenberg, and Teresa Anna Steiner; licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 14; pp. 14:1–14:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

The classic string indexing problem is to preprocess a string  $S$  into a compact data structure that supports efficient subsequent pattern matching queries, that is, given a pattern string  $P$ , report all occurrences of  $P$  within  $S$ . An *occurrence* of  $P$  within  $S$  is an index  $i$ ,  $0 \leq i < |S|$ , such that  $P = S[i \dots i + |P| - 1]$ . In this paper, we introduce a basic extension of string indexing, where the goal is to report *consecutive occurrences* of the pattern  $P$  that occur *close* to each other in  $S$ . Here, a consecutive occurrence is a pair  $(i, j)$ ,  $i < j$ , such that  $P$  occurs at positions  $i$  and  $j$  in  $S$  and there is no occurrence of  $P$  between  $i$  and  $j$ , and close to each other means that the distance  $j - i$  between the occurrences should be small. More precisely, given a pattern  $P$  and an integer parameter  $k > 0$ , define the *top- $k$  close consecutive occurrences* of  $P$  to be the  $k$  consecutive occurrences of  $P$  in  $S$  with the smallest distances. Given a string  $S$  the *string indexing for top- $k$  close consecutive occurrences* (SITCCO) problem is to preprocess  $S$  into a data structure that supports top- $k$  close consecutive occurrences queries. The goal is to obtain a compact data structure while supporting fast queries in terms of the length of the pattern  $P$  and the number of reported occurrences  $k$ . For an example, see Figure 1.

$P = \text{AN}$   
 $S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS}$   
 0 5 10 15 20 25 30 35 40

■ **Figure 1**  $P$  occurs at positions 4, 7, 11, 22, 24, 26, 30, 39 and 41 in  $S$ . The top 5 close consecutive occurrences are (22, 24), (24, 26), (39, 41), (4, 7), and (7, 11), with the tie between (7, 11) and (26, 30) broken arbitrarily.

Surprisingly, the SITCCO problem has not been studied before even though it is a natural variant of string indexing and several closely related problems have been extensively studied (see related work below).

### 1.1 Results and Techniques

To state the complexity bounds, let  $n$  and  $m$  denote the lengths of  $S$  and  $P$ , respectively. An immediate approach to solve the SITCCO problem is to store the suffix tree of  $S$  using  $O(n)$  space. To answer a query on  $P$  with parameter  $k$ , we traverse the suffix tree to find *all* occurrences of  $P$ , construct the consecutive occurrences, and then sort these to output the top- $k$  close consecutive occurrences. Naively, this requires two sorts of size  $\text{occ}$ , where  $\text{occ}$  is the total number of occurrences of  $P$ , giving a query time of  $O(m + \text{occ} \log \text{occ})$ . Using more advanced data structures [9, 10], the query time can be reduced to  $O(m + \text{occ})$  while still using linear space. Note that  $\text{occ}$  can be much larger than  $k$ . Alternatively, we can store at every node in the suffix tree the set of all consecutive occurrences sorted by distance using  $O(n^2)$  space. To answer a query we find the node corresponding to  $P$  and simply report the first  $k$  of the stored consecutive occurrences in optimal  $O(m + k)$  time.

To achieve better trade-offs, one might try to use a strategy similar to range minimum query (RMQ), where the ranges are subsequent ranges in the suffix array and the values are distances between pairs of suffix indexes in  $S$ . However, there are several problems with that idea: first, there are  $\Theta(|S|^2)$  possible pairs of suffix indexes within  $S$ , and it is not immediately clear how many of them can correspond to *consecutive* occurrences of a pattern (our arguments

from Section 3 imply that this number is bounded by  $O(n \log n)$ ). Secondly, when taking the union of two ranges, the set of closest (consecutive) pairs can change completely: consider for example the string  $S = A B A C A B A C D A B D A C D A B D A C$ . While the string  $A$  has occurrences  $\{0, 2, 4, 6, 9, 12, 15, 18\}$ , the string  $AB$  has occurrences  $\{0, 4, 9, 15\}$  and  $AC$  has  $\{2, 6, 12, 18\}$ . Note that for  $P = A$ , the top-3 consecutive occurrences are  $(0, 2)$ ,  $(2, 4)$  and  $(4, 6)$ , while for  $AB$  they are  $(0, 4)$ ,  $(4, 9)$  and  $(9, 15)$  and for  $AC$  they are  $(2, 6)$ ,  $(6, 12)$  and  $(12, 18)$ . Both the pairs and the distances are completely different between  $A$  and its extensions. Thus, there is an issue of non-decomposability, which is a main challenge in this particular problem. However, in the rest of our paper we will show that we can use suffix tree decompositions and amortized arguments to bound the number of changes that can happen in the set of consecutive occurrences of substrings corresponding to positions on some paths in the suffix tree.

We obtain the following significantly improved time-space trade-offs:

► **Theorem 1.** *Given a string  $S$  of length  $n$  and  $\epsilon$ ,  $0 < \epsilon \leq 1$ , we can build a data structure that can answer top- $k$  close consecutive occurrences queries using either*

- (i)  $O(n \log n)$  space and  $O(m + k)$  query time or
- (ii)  $O(\frac{n}{\epsilon})$  space and  $O(m + k^{1+\epsilon})$  query time.

Here,  $m$  is the length of the query pattern.

Hence, Theorem 1(i) achieves optimal query time using near-linear space. Alternatively, Theorem 1(ii) achieves linear space, for constant  $\epsilon$ , while supporting queries in near-optimal  $O(m + k^{1+\epsilon})$  time.

To achieve Theorem 1 we develop several data structural techniques that may be of independent interest. First, we translate the problem into a line segment intersection problem on the heavy path decomposition of the suffix tree. This leads to the  $O(n \log n)$  space and optimal query time bound of Theorem 1(i). We note that Navarro and Thankachan [31] used similar techniques for a closely related problem (see related work below). To reduce space, we introduce a novel recursive clustering method on trees. The decomposition partitions the tree into a hierarchy of depth  $O(\log \log n)$  consisting of subtrees of doubly exponentially decreasing sizes. We show how to combine the decomposition with the techniques of the simple algorithm from Theorem 1(i) to obtain an  $O(n \log \log n)$  space and  $O(m + k^{1+\epsilon})$  query time solution. Finally, we show how to efficiently compress the hierarchy of data structures into rank space leading to the linear space and  $O(m + k^{1+\epsilon})$  query time bound of Theorem 1(ii).

We apply these techniques to three related problems: Firstly, we address the natural “opposite” problem of reporting the  $k$  consecutive occurrences of *largest* distance, which can be solved using similar but not identical techniques. Secondly, we apply our framework to the related problem of reporting consecutive occurrences with distances within a specified interval, considered by Navarro and Thankachan [31], and give an improvement for a special case. Finally, we show how this allows us to efficiently report all non-overlapping consecutive occurrences of a pattern. The proofs for these extensions are deferred to the full version of the paper.

## 1.2 Related Work

To the best of our knowledge, the SITCCO problem has not been studied before, even though distances between occurrences is a natural extension for string indexing and several related problems have been studied extensively.

A closely related problem was considered by Navarro and Thankachan [31], who showed how to efficiently report consecutive occurrences with distances within a specified interval. They gave an  $O(n \log n)$  space and  $O(m + \text{occ})$  time solution, where  $\text{occ}$  denotes the number of reported consecutive occurrences. We note that their result can be adapted to the `SITCCO` problem to achieve the same bounds as in Theorem 1(i). However, our solution is simpler and does not rely on heavy word RAM techniques such as persistent van Emde Boas trees [12]. Our techniques can also be used to solve the problem considered by Navarro and Thankachan getting the same space and time bounds as they obtain, and we can achieve improved bounds in a special case (see Section 7).

A lot of work has been done on the related problem of string indexing for patterns under various *distance constraints*, where the goal is to report occurrences of (one or more) patterns that are within a given distance or interval of distances of each other [4, 6, 7, 11, 23, 24, 25]. An important difference between those works and our work is that all those solutions use time proportional to *all* pairs of occurrences with distances in the given range, in contrast to only finding *consecutive* occurrences. Note that if the goal is to find occurrences of a given maximal distance, one can find the close *consecutive* occurrences first and then construct all pairs satisfying the constraint.

Another line of related work is indexing collections of strings, called *documents*. Here the goal is to find documents containing patterns subject to various constraints. For a comprehensive overview see the survey by Navarro [28]. Several results on supporting efficient top- $k$  queries are known [8, 18, 19, 20, 20, 21, 26, 27, 29, 30, 32, 34]. In this context the goal is to efficiently report the  $k$  documents of smallest weight. The weights can depend on the query and can be the distance between the closest pair of occurrences of a given pattern [20, 20, 26, 29, 29, 32]. The problem can be solved in linear space and optimal  $O(k)$  time, in addition to finding the locus of the pattern in the suffix tree [32]. While this problem statement resembles ours, there is no direct translation from those results to our problem, since the documents are considered individually, and for a single document only the pair of occurrences with minimum distance within the document is considered.

### 1.3 Outline

The paper is organized as follows. In Section 2 we introduce some notation and recall results on string indexing. In Section 3 we build a simple data structure and prove Theorem 1(i). In Section 4 we recall a method for tree clustering and show how to use it to solve a simplified version of the problem. In Section 5 we introduce a recursive clustering method that allows us to use the ideas from Section 4 on the actual problem. This gives an  $O(n \log \log n)$  space and  $O(m + k^2)$  time data structure. In Section 6, we show how to reduce the space to linear while achieving the same query time, and then generalize the recursion to get Theorem 1(ii) for any  $0 < \epsilon \leq 1$ . Finally, in Section 7 we apply our techniques to related problems.

## 2 Preliminaries

We introduce some notation and recall basic results from string indexing.

A *string*  $S$  of length  $n$  is a sequence  $S[0]S[1] \dots S[n-1]$  of characters from an alphabet  $\Sigma$ . A contiguous subsequence  $S[i, j] = S[i]S[i+1] \dots S[j-1]$  is a *substring* of  $S$ . The substrings of the form  $S[i, n]$  are the *suffixes* of  $S$ .

The *suffix tree* [35] is a compact trie of all suffixes of  $S\$$ , where  $\$$  is a symbol not in the alphabet, and is lexicographically smaller than any letter in the alphabet. Using perfect hashing [16], it can be stored in  $O(n)$  space and solve the string indexing problem (i.e., find

and report all occurrences of a pattern  $P$  in  $O(m + \text{occ})$  time, where  $m$  is the length of  $P$  and  $\text{occ}$  is the number of times  $P$  occurs in  $S$ . The *suffix array* stores the suffix indices of  $S$  in lexicographic order. The suffix tree has the property that the leaves below any node represent suffixes that appear in consecutive order in the suffix array. Brodal et al. [10] show that there is a linear space data structure that allows outputting all entries within a given range of an array in sorted order using time linear in size of the output. This data structure on the suffix array together with the suffix tree can output all occurrences of a pattern *sorted by text order* in  $O(n)$  space and  $O(m + \text{occ})$  time.

For any node  $v$  in the suffix tree, we define  $\text{str}(v)$  to be the string found by concatenating all labels on the path from the root to  $v$ . The *locus* of a string  $P$ , denoted  $\text{locus}(P)$ , is the minimum depth node  $v$  such that  $P$  is a prefix of  $\text{str}(v)$ .

### 3 A Simple $O(n \log n)$ Space Solution

In this section, we present a simple solution that solves the SITCCO problem in  $O(n \log n)$  space and  $O(m + k)$  query time. This solution will be a key component in our more advanced structures in the following sections. We note that the results by Navarro and Thankachan [31] for the related problem of reporting consecutive occurrences with distances within a specified interval can be modified to achieve the same complexities. However, our solution is simpler and does not rely on heavy word RAM techniques such as persistent van Emde Boas trees [12].

Let  $D(v)$  denote the set of consecutive occurrences of  $\text{str}(v)$ . Naively, if we store for each node  $v$  the set  $D(v)$  in sorted order, we can directly answer a query for the top- $k$  close consecutive occurrences of a pattern  $P$  by reporting the  $k$  smallest elements in  $D(\text{locus}(P))$ . This solves the problem in  $O(n^2)$  space and  $O(m + k)$  query time. The main idea in our simple solution is to build a heavy path decomposition of the suffix tree and compactly represent sets on the same path via a reduction to the orthogonal line segment intersection problem while maintaining optimal time queries. This is similar to the data structure by Navarro and Thankachan [31], but our reduction is different.

#### Heavy path decomposition

A *heavy path decomposition* of a tree  $T$  is defined as follows: Starting from the root, at every node, we choose the edge to the child with the largest subtree as *heavy edge*, until we reach a leaf. Ties are broken arbitrarily. This defines a heavy path, and all edges hanging off the heavy path are *light edges*. The root of a heavy path  $h$  is called the *apex* of the path, denoted  $\text{apex}(h)$ . We then recursively decompose all subtrees hanging off the path. The heavy path decomposition has the following property:

► **Lemma 2** (Sleator and Tarjan [33]). *Given a tree  $T$  of size  $n$  and a heavy path decomposition of  $T$ , any root-to-leaf path in  $T$  contains at most  $O(\log n)$  light edges.*

#### Orthogonal line segment intersection

Similarly as Navarro and Thankachan [31], we are going to reduce the problem to a geometric problem on orthogonal line segment intersection. Specifically, we are going to reduce to the following problem: Let  $L$  be a set of  $n$  vertical line segments in a plane with non-negative  $x$ -coordinates. The *orthogonal line segment intersection problem* is to preprocess  $L$  to support the query:

- **smallest-segments**( $y_0, k$ ): return the first  $k$  segments intersecting the horizontal line with  $y$ -coordinate  $y_0$  in left-to-right order.

We will assume that  $y_0$  is an integer, which suffices for our purpose. Let  $N$  be the maximum  $y$ -coordinate of a segment in  $L$ . The following lemma follows easily from the results on partially persistent data structures by Driscoll et al. [14].

► **Lemma 3.** *We can solve the line segment intersection problem as described above in  $O(n + N)$  space and  $O(k)$  time.*

**Proof.** Consider the  $x$ -coordinates as the elements of a set  $X$  and the  $y$ -coordinate as time. The version of  $X$  at a time  $y_0$  contains exactly the  $x$ -coordinates of the line segments which intersect the horizontal line at  $y_0$ . Now, the data structure is a partially persistent sorted doubly linked list  $L$  on the elements of  $X$ . The elements are sorted in increasing order. Since we have at most  $n$  line segments, the maximum size of  $X$  as well as the maximum number of updates is  $n$ . Each update changes only  $O(1)$  pointers in the linked list. Using the node copying technique from Driscoll et al. [14] we can build a partially persistent linked list using  $O(n)$  space. To be able to find version  $y_0$  in constant time, we keep an array of size  $N$  with a pointer to the root of the version at each possible time step. For a query  $(y_0, k)$ , use the sorted linked list  $L$  to report the  $k$  smallest elements at time  $y_0$ .

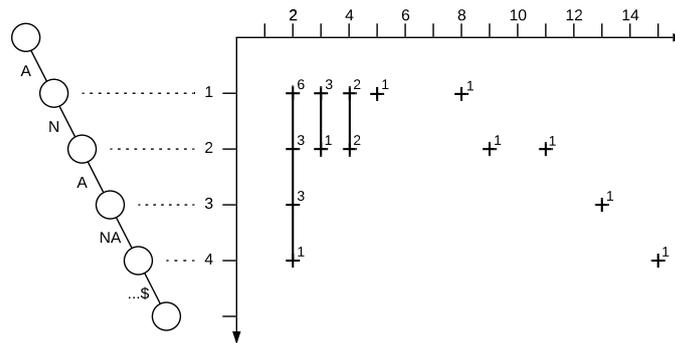
If we use a linear scan to find the place to insert an element or find the element to be deleted we get a preprocessing time of  $O(n^2)$ . This can be improved to  $O(n \log n)$  by using a (non-persistent) balanced binary search tree during the preprocessing holding all elements in the current version of  $L$  together with a pointer to their node in the current version. When performing an update the binary search tree is used to find the position where the element must be inserted/deleted in  $O(\log n)$  time. After the preprocessing step the tree is discarded. ◀

### 3.1 Data Structure

We construct a heavy path decomposition of the suffix tree  $T$  of  $S$ . Our data structure consists of a line segment data structure from Lemma 3 for each heavy path of  $T$  that compactly encodes the sets  $D(v)$  for each node  $v$  on the path.

We describe the contents of the data structure for a single heavy path  $h = v_1, \dots, v_\ell$ , where  $v_1$  is the apex of the path. Consider a consecutive occurrence  $(i, j)$  on some node on  $h$  and imagine moving down the heavy path from top to bottom. Either  $(i, j)$  is a consecutive occurrence at the apex of  $h$  or it will become a consecutive occurrence as soon as every suffix starting at an index between  $i$  and  $j$  has branched off the heavy path. Then it will stay a consecutive occurrence until either the suffix corresponding to  $i$  or the suffix corresponding to  $j$  (or both) branch off  $h$ . Thus, there exists an interval  $[d_1, d_2]$  of depths on the heavy path such that  $(i, j) \in D(v_d)$  if and only if  $d \in [d_1, d_2]$ . We say that  $(i, j)$  is *alive* in this interval.

We encode the consecutive occurrences by line segments in the plane which describe their distance and the interval in which they are alive along the heavy path. Conceptually, the  $x$ -coordinate in our coordinate system corresponds to the distance of a consecutive pair, and the  $y$ -coordinate corresponds to the depth on the heavy path. Now, for each consecutive occurrence  $(i, j)$ , we define a vertical line segment with  $x$ -coordinate set to its distance, and  $y$ -coordinate spanning the interval  $[d_1, d_2]$ , where  $[d_1, d_2]$  is the interval in which  $(i, j)$  is alive. For an example, see Figure 2. Our data structure for  $h$  stores the above line segments in the line segment data structure from Lemma 3. For each line segment in the data structure we store a pointer to the pair of occurrences it represents. The full data structure for  $T$  consists of the line segment data structures for all of the heavy paths in  $T$ .



■ **Figure 2** Line segments for a heavy path from the suffix tree for “BATMAN-AND-ANNA-SING-NANANANA-AND-EAT-BANANAS”. Here, if we have overlapping line segments, we denote by a number how many consecutive occurrences the current segment corresponds to. At depth 1, we have a line segment corresponding to pairs of consecutive occurrences of string A - there are six pairs that have a distance of 2, three pairs that have a distance of 3, two pairs that have a distance of 4, and so on. At depth 2, we encode the consecutive occurrences of string AN. Some of them are the same as for string A.

### Space analysis

For a given heavy path  $h$ , a leaf in the subtree of  $\text{apex}(h)$  can be in at most two consecutive occurrences in  $D(\text{apex}(h))$ . Consider a light edge  $(v_d, u)$  leaving  $h$  at depth  $d$ . Any leaf in the subtree rooted at  $u$  can be part of at most two consecutive occurrences in  $D(v_d)$ . A single leaf can thus make at most two consecutive occurrences from  $D(v_d)$  disappear in  $D(v_{d+1})$  and at most one new consecutive occurrence appear. If we consider all leaves that leave  $h$ , we therefore get at most three changes per leaf. Thus, for a given heavy path  $h$  a leaf in the subtree of  $\text{apex}(h)$  can be in at most two consecutive occurrences in  $D(\text{apex}(h))$  and can cause at most three changes of line segments in the line segment data structure for  $h$ . Since any root-to-leaf path can intersect at most  $\log n$  heavy paths, any leaf can contribute  $O(\log n)$  line segments. Overall, this means that there are at most  $O(n \log n)$  line segments in total. For a single heavy path  $h$  the line segment data structure from Lemma 3 uses linear space in the number of segments and the length of  $h$ . The sum of the lengths of the heavy paths is  $O(n)$ , since the heavy paths are disjoint. Thus the total space usage is  $O(n \log n)$ .

### 3.2 Algorithm

Given a pattern  $P$  and an integer  $k$  we can now answer a query as follows. We begin by finding  $\text{locus}(P)$  in the suffix tree. Let  $h$  be the heavy path that the locus is on and let  $d_P$  be the depth of  $\text{locus}(P)$  on  $h$ . We do a  $\text{smallest-segments}(d_P, k)$  query on the line segment data structure stored for  $h$  and report the consecutive occurrences corresponding to the returned line segments.

#### Correctness

By definition,  $D(\text{locus}(P))$  contains the consecutive occurrences of  $P$ . Thus, every consecutive occurrence of  $P$  defines a line segment in the data structure for  $h$  and the horizontal line with  $y$ -coordinate set to  $d_P$  intersects exactly those line segments. Since we set the  $x$ -coordinate of every line segment to the distance of its consecutive occurrence, the line segments are sorted left-to-right by increasing distance. Thus, the first  $k$  line segments intersecting the horizontal line at  $y = d_P$  correspond to the top- $k$  close consecutive occurrences.

### Time analysis

The time for finding  $\text{locus}(P)$  in the suffix tree is  $O(m)$ . The time for querying the line segment data structure from Lemma 3 is  $O(k)$ , so the total time complexity is  $O(m + k)$ . This proves Theorem 1(i).

## 4 A Linear Space Solution for Fixed $k$

In this section, we present a linear space and  $O(m + k)$  time solution for the simpler problem where  $k$  is known at construction time. That is, given a string  $S$  and a positive integer  $k$ , we preprocess  $S$  into a compact data structure such that given a pattern string  $P$ , we can efficiently find the top- $k$  close consecutive occurrences of  $P$  in  $S$ . This data structure demonstrates one of the key ideas that our final result builds on.

The main idea behind the data structure is to store the line segment solution from Section 3 for some path segments of the suffix tree, such that all nodes that are not on these paths are within small subtrees. For nodes within such small subtrees we can find all consecutive occurrences without spending too much time. Specifically, we will partition the suffix tree into *clusters*, satisfying some properties. We are going to define this cluster partition next.

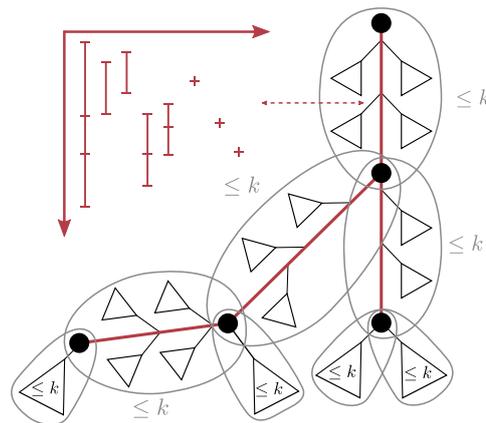
### 4.1 Cluster Partition

For a connected subgraph  $C \subseteq T$ , a *boundary node*  $v$  is a node  $v \in C$  such that either  $v$  is the root of  $T$ , or  $v$  has an edge leaving  $C$  – that is, there exists an edge  $(v, u)$  in the tree  $T$  such that  $u \in T \setminus C$ . A *cluster* is a connected subgraph  $C$  of  $T$  with at most two boundary nodes. A cluster with one boundary node is called a *leaf cluster*. A cluster with two boundary nodes is called a *path cluster*. For a path cluster  $C$ , the two boundary nodes are connected by a unique path. We call this path the *spine* of  $C$ . A *cluster partition* is a partition of  $T$  into clusters, i.e. a set  $CP$  of clusters such that  $\bigcup_{C \in CP} V(C) = V(T)$  and  $\bigcup_{C \in CP} E(C) = E(T)$  and no two clusters in  $CP$  share any edges. Here,  $E(G)$  and  $V(G)$  denote the edge and vertex set of a (sub)graph  $G$ , respectively. We need the next lemma which follows from well-known tree decompositions [1, 2, 3, 15] (see Bille and Gørtz [5] for a direct proof).

► **Lemma 4.** *Given a tree  $T$  with  $n$  nodes and a parameter  $\tau$ , there exists a cluster partition  $CP$  such that  $|CP| = O(n/\tau)$  and every  $C \in CP$  has at most  $\tau$  nodes. Furthermore, such a partition can be computed in  $O(n)$  time.*

### 4.2 Data Structure

For the suffix tree of  $S$ , we build a clustering as in Lemma 4 with parameter  $\tau$  set to  $k$  to get  $O(n/k)$  clusters of size at most  $k$ . For the spine of every path cluster, we build a line segment data structure similar to the one from Section 3. The difference is that for any depth, we only maintain the line segments that correspond to the top- $k$  close consecutive occurrences for that depth. Let  $v_1, \dots, v_l$  denote the nodes on the spine, starting at the top boundary node. Note that for any consecutive occurrence that appears for the first time in  $D(v_{d+1})$  there is a consecutive occurrence in  $D(v_d)$  of smaller distance which is no longer present in  $D(v_{d+1})$ . It follows that, when moving down the spine, once a consecutive occurrence  $(i, j)$  is amongst the  $k$  closest, it will stay amongst the  $k$  closest until suffix  $i$  or  $j$  branches off the spine. Thus, there exists an interval  $[d_1, d_2]$  of consecutive depths such that  $(i, j)$  is amongst the  $k$  closest pairs in  $D(v_d)$  if and only if  $d \in [d_1, d_2]$ . For a consecutive occurrence  $(i, j)$  that is amongst



■ **Figure 3** The suffix tree is divided into clusters (grey loops) of size  $\leq k$  which are either leaf clusters, or path clusters with spines marked in red. For every spine we store a line segment data structure, also marked in red.

the  $k$  closest for any  $v$  on the spine, we define a line segment where the  $x$ -coordinate is its distance and the  $y$ -coordinate is spanning the interval  $[d_1, d_2]$ , where  $[d_1, d_2]$  is the interval in which  $(i, j)$  is amongst the  $k$  closest pairs. For these line segments we store the data structure from Lemma 3. Again, for each line segment we store the pair of occurrences it represents. We store this data structure for the spine of each cluster and for every node that is on that spine we store a pointer to the data structure. For boundary nodes that are on multiple spines we store a pointer to any one of them. See Figure 3 for an illustration of this structure. Additionally we store the suffix array and the sorted range reporting data structure of Brodal et al. [10] on the suffix array.

### Space analysis

We show that for every path cluster there are  $O(k)$  line segments: We still have the property that a line segment only ends if a corresponding leaf branches off the spine. In that case, it might be replaced either by a new consecutive occurrence or by a consecutive occurrence that was there before but was not amongst the  $k$  closest. Note that at any node on the spine except the boundary nodes, any subtrees branching off the spine are fully contained within the cluster, and as such have total size at most  $k$ . Between the top boundary node and the next node on the spine, we have no bound as to how many leaves can branch off – however, since we only store line segments corresponding to the top- $k$  consecutive occurrences, at most  $k$  line segments can be replaced by  $k$  other line segments. For the rest of the spine, at most  $k$  leaves can branch off in total. Every leaf that branches off can cause at most two line segments to end and two new line segments to begin. As such there can be at most  $O(k)$  line segments. As the size of the line segment data structure is linear in the number of line segments and in the length of the spine, any line segment data structure of a path cluster uses  $O(k)$  space. As both the sorted range reporting data structure and the suffix array have linear space complexity, the complete data structure occupies  $O((n/k)k + n) = O(n)$  space.

### 4.3 Algorithm

Given a pattern  $P$  we can now answer the top- $k$  query. We begin by finding  $\text{locus}(P)$  in the suffix tree. If the locus is on a spine, we query the line segment data structure for that spine. Otherwise the locus is either in a subtree hanging off a spine or in a leaf cluster. In

## 14:10 String Indexing for Top- $k$ Close Consecutive Occurrences

both cases, there are at most  $k$  occurrences of our pattern  $P$ . We find all occurrences of  $P$  in text order, using the sorted range reporting data structure. This allows us to report the consecutive occurrences: Let  $i_1, \dots, i_l$  denote the leaves in text order, then the consecutive occurrences are  $(i_1, i_2), (i_2, i_3), \dots, (i_{l-1}, i_l)$ . Note that  $l \leq k$ , since the size of the subtree is at most  $k$ .

### Correctness

By construction, for any depth on a spine, the top- $k$  close consecutive occurrences of the corresponding substring will have corresponding line segments present at that depth in the line segment data structure. If the locus is on a spine, then by the arguments in Section 3, the line segment data structure will report the top- $k$  close consecutive occurrences. If the locus is not on a spine, then there are at most  $k$  occurrences of  $P$  in total, since any subtree hanging off a spine and any leaf cluster has at most  $k$  leaves. Thus, by constructing and reporting all consecutive occurrences of  $P$  we report the top- $k$  close consecutive occurrences.

### Time analysis

We find the locus in  $O(m)$  time. If we land on a spine we report in  $O(k)$  time. Otherwise, we are in a subtree of size at most  $O(k)$  and thus  $P$  has at most  $k$  occurrences. Using sorted range reporting we can find the occurrences in text order using  $O(k)$  time. The total time for a query is thus  $O(m + k)$ .

We are going to use this data structure with different parameters in Section 5. For a general parameter  $\tau$ , we have the following lemma:

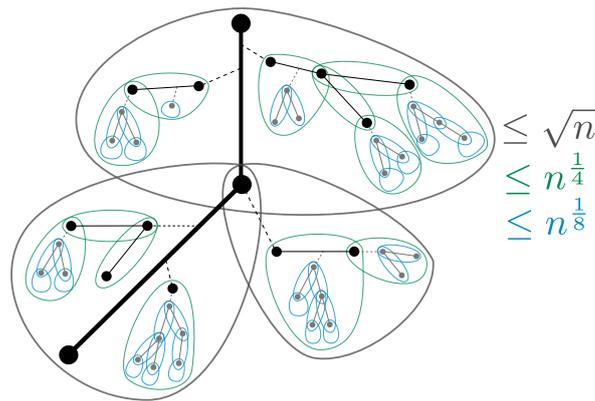
► **Lemma 5.** *For any positive integer  $\tau$ , there exists a cluster partition of the suffix tree and a linear space data structure with the following properties:*

1. *For any  $k \leq \tau$  and  $P$  such that  $\text{locus}(P)$  is on the spine of a cluster, we can report the top- $k$  close consecutive occurrences in  $O(m + k)$  time.*
2. *For any  $P$  such that  $\text{locus}(P)$  is not on a spine, we can report the top- $k$  close consecutive occurrences in  $O(m + \tau)$  time.*

**Proof.** We build the data structure described in this section for parameter  $\tau$  taking the role of  $k$ . In case 1, we query the line segment data structure for the depth of  $\text{locus}(P)$  on the path and  $k$ . Since  $k \leq \tau$  this will correctly output the top- $k$  close consecutive occurrences of  $P$ . In case 2, we have shown that we can construct the top- $\tau$  close consecutive occurrences. Using the linear time selection algorithm by Blum et al. [9] we can find the top- $k$  of those: We use the algorithm to find the consecutive occurrence of  $k^{\text{th}}$  smallest distance  $d$ ; then we traverse all the consecutive occurrences and output those of distance  $\leq d$ . If needed, we crop the output to report no more than  $k$  consecutive pairs. ◀

## 5 An $O(n \log \log n)$ Space Solution for General $k$

We now show how to leverage the solution from Lemma 5 to obtain a data structure that can answer queries for any  $k$ . The idea is to recursively cluster the suffix tree, such that we always either land on a spine with a sufficient number of consecutive occurrences stored, or in a sufficiently small subtree.



■ **Figure 4** Here, we see the recursive clustering: The black clustering is the coarsest clustering and the green and blue are finer sub-clusters.

## 5.1 Data Structure

Our data structure consists of the suffix tree decomposed into clusters of decreasing size, with the line segment data structure stored for every spine as before. We build it in the following way. First we build the solution from Lemma 5 with parameter  $\tau_1 = \sqrt{n}$ , resulting in clusters of size at most  $\sqrt{n}$ . For every subtree hanging off a spine and every leaf cluster, we apply the solution with parameter  $\tau_2 = \sqrt{\tau_1}$ . We keep recursively applying the solution with parameter  $\tau_i = \sqrt{\tau_{i-1}}$  until reaching a constant cluster size. For notational convenience, additionally define  $\tau_0 = n$ . See Figure 4 for an illustration of this data structure. Again we additionally store the suffix array and the sorted range reporting data structure of Brodal et al. [10] on the suffix array.

### Space analysis

The suffix array and sorted range reporting structure occupy  $O(n)$  space. For a tree of size  $\tilde{n}$  and any  $\tau$ , the data structure from Lemma 5 uses at most  $O(\tilde{n})$  space. Since at every recursion level, we build the data structure from Lemma 5 on non-overlapping subtrees of the suffix tree, every recursion level uses at most  $O(n)$  space. As the cluster size at every level of recursion is the square root of the previous cluster size, there are at most  $O(\log \log n)$  levels. The complete data structure thus uses  $O(n \log \log n)$  space.

## 5.2 Algorithm

Given a query with pattern  $P$  and parameter  $k$ , we can now answer in the following way. As before, we begin by finding the locus of the pattern in the suffix tree. This node is now either on the spine of some cluster or in a cluster of constant size. If it is on the spine of a cluster of size  $\tau_i$ , and if  $k \leq \tau_i$ , then we query the line segment data structure for that spine, which allows us to report the top- $k$  close consecutive occurrences. Otherwise, we find all occurrences of  $P$  and construct the top- $k$  close consecutive occurrences by using linear time selection as in the proof of Lemma 5.

### Correctness

The correctness of the algorithm follows by the same arguments as previous sections.

**Time analysis**

Finding the locus in the suffix tree takes  $O(m)$  time. The locus is either on the spine of a cluster, or within a cluster of constant size. In a constant sized cluster, clearly we can do all operations described above in constant time. If the locus is on the spine of a cluster with parameter  $\tau_i$ , and  $k \leq \tau_i$ , then we are in case 1 of Lemma 5 with  $\tau = \tau_i$  and can report the top- $k$  close consecutive occurrences using a total of  $O(m + k)$  time. If  $k > \tau_i$ , then we are in case 2 of Lemma 5 with  $\tau = \tau_{i-1}$ . Note that  $\tau_{i-1} = \tau_i^2 < k^2$ . Therefore, we can find the top- $k$  close consecutive occurrences in  $O(m + \tau_{i-1}) = O(m + k^2)$  time. In total, the worst case query time is then  $O(m + k^2)$ . In summary, this gives the following result:

► **Lemma 6.** *Given a string  $S$  of length  $n$ , we can build a data structure that can answer top- $k$  close consecutive occurrences queries using  $O(n \log \log n)$  space and  $O(m + k^2)$  query time. Here,  $m$  is the length of the query pattern.*

**6 A Linear Space Solution**

We now show how to reduce the space consumption of the solution presented in Section 5. Observe that in any cluster of level  $i$ , we only have  $O(\tau_i)$  objects. If we can reduce all objects within a cluster to a “universe size” of  $O(\tau_i)$  instead of  $O(n)$ , we can use  $O(\tau_i \log \tau_i)$  bits instead of  $O(\tau_i \log n)$  bits per cluster. In the following, consider a cluster  $C$  of level  $i$ .

**Reducing the line segment data structure**

In the line segment data structure for cluster  $C$ , by the analysis of previous sections, there are at most  $O(\tau_i)$  line segments and  $\tau_i$  different depths on the path. Let  $c$  be a constant such that there are at most  $c\tau_i$  line segments for each cluster. We map every unique  $x$ -coordinate of a line segment to a unique element in  $\{1, \dots, c\tau_i\}$  in a way that preserves order. That is, map the minimum  $x$ -coordinate to 1, the smallest  $x$ -coordinate that is bigger than the minimum to 2, and so on. This gives us a modified line segment data structure that preserves the properties we need but is restricted to a  $c\tau_i \times \tau_i$  grid.

**Reducing the leaf pointers**

For any line segment, we have to store pointers that allow us to report the corresponding pair of consecutive occurrences. Doing so naively uses  $2 \log n$  bits per line segment. In the following, we show how to reduce that to  $4 \log \tau_i$ , for a cluster  $C$  of level  $i$ . The idea is to store the offset within the suffix array range defined by the top boundary node  $r$  of  $C$ . More precisely, let  $[a_r, b_r]$  be the range in the suffix array spanning the leaves below  $r$ . Then for any leaf  $l$  in the subtree rooted at  $r$  define  $\text{off}(l) = SA^{-1}(l) - a_r$ . By the way our recursion is defined,  $C$  is fully contained in a subtree of size at most  $\tau_i^2$ , and thus  $r$  has at most  $\tau_i^2$  leaves below it. It follows that for any leaf  $l$  in the subtree of  $r$ ,  $\text{off}(l)$  is a number between in  $[0, \tau_i^2 - 1]$  and can be stored using  $2 \lceil \log \tau_i \rceil$  bits.

**6.1 Data Structure**

Our data structure is now defined as follows: We have a clustering of the suffix tree as in Section 5. For every spine on level  $i$ , we store the line segment data structure reduced to a  $c\tau_i \times \tau_i$  grid. Every line segment corresponding to a pair  $(i, j)$  stores the pair  $(\text{off}(i), \text{off}(j))$  as additional information. For every node on the spine, we store a pointer to the spine data structure and to the top boundary node of the spine. Additionally, we store the suffix array and the sorted range reporting structure, as well as two integers for every node in the suffix tree, that define the range of leaves below the node in the suffix array.

### Space analysis

The suffix array and the sorted range reporting data structure use space  $O(n)$ . Storing the range in the suffix array plus at most two pointers per node uses  $O(n)$  space. For a cluster  $C$  of level  $i$ , we store the line segment data structure from Lemma 3 for a  $c\tau_i \times \tau_i$  grid. Since the data structure from Lemma 3 works in the word RAM model (as do all data structures presented in this paper), we can store the data structure using  $O(\tau_i \log \tau_i)$  bits. For each of the at most  $c\tau_i$  line segments we store  $4 \log \tau_i$  bits for the encoding of the consecutive pair. Thus, we can store the data structure for cluster  $C$  using  $O(\tau_i \log \tau_i)$  bits. As in the previous section, at every recursion level, we cluster non-overlapping subtrees. The reduced cluster solution of a subtree of size  $\tilde{n}$  with parameter  $\tau_i$  uses  $O(\frac{\tilde{n}}{\tau_i} \tau_i \log \tau_i) = O(\tilde{n} \log \tau_i)$  bits. The total space for all clusters of level  $i$  thus becomes  $O(n \log \tau_i)$ . Summing over all recursion levels, we get

$$\sum_{i=0}^{\lceil \log \log n \rceil} O(n \log \tau_i) = \sum_{i=0}^{\lceil \log \log n \rceil} O\left(n \log n^{(1/2^i)}\right) = \sum_{i=0}^{\lceil \log \log n \rceil} \frac{1}{2^i} O(n \log n) = O(n \log n) \text{ bits,}$$

that is,  $O(n)$  words.

## 6.2 Algorithm

We query the data structure as follows: If we land on a spine and  $k \leq \tau_i$ , we query the line segment data structure and get  $k$  pairs of the form  $(\text{off}(i), \text{off}(j))$ . We then use the pointer to get to the root of the spine and use the range in the suffix array to translate each encoding back to the original suffix number, using constant time per leaf. Otherwise, we proceed as described in Section 5. Since the decoding can be done in constant time per leaf, the time complexities are the same as in Section 5. We have shown the following result:

► **Lemma 7.** *Given a string  $S$  of length  $n$ , we can build a data structure that can answer top- $k$  close consecutive occurrences queries using  $O(n)$  space and  $O(m + k^2)$  query time. Here,  $m$  is the length of the query pattern.*

In order to get Theorem 1(ii), we cluster according to a parameter  $\epsilon$ ,  $0 < \epsilon \leq 1$ , using the following recursion:

$$\tau_0 = n \quad \text{and} \quad \tau_i = \tau_{i-1}^{\frac{1}{1+\epsilon}}.$$

Hence, the total space in bits is now:

$$\sum_{i=0}^{\lceil \log_{1+\epsilon} \log n \rceil} O\left(n \log n^{1/(1+\epsilon)^i}\right) = \sum_{i=0}^{\infty} \left(\frac{1}{1+\epsilon}\right)^i O(n \log n) = \left(1 + \frac{1}{\epsilon}\right) O(n \log n),$$

that is,  $O\left(\frac{n}{\epsilon} \log n\right)$  bits, so  $O\left(\frac{n}{\epsilon}\right)$  words. For the query time, there are again two cases. In the case where  $\text{locus}(P)$  is on a spine with  $k \leq \tau_i$ , we get optimal  $O(m + k)$  time, as before. For the other case, we have at most  $\tau_{i-1} = \tau_i^{1+\epsilon} < k^{1+\epsilon}$  occurrences of  $P$ , which gives us a time complexity of  $O(m + k^{1+\epsilon})$ . This concludes the proof of the main result.

## 7 Extensions

Our results can be extended to a couple of related problems. We give an overview here and refer to the full version for details and proofs. In Section 7.1, we introduce the “opposite” problem of reporting the  $k$  consecutive occurrences of *largest* distance. The extension is

quite natural, though it does require some careful analysis. In Section 7.2, we then relate the results from Section 7.1 and the solutions to SITCCO to the problem of finding consecutive occurrences with distances in a specified interval, considered by Navarro and Thankachan [31]. Our results give improved complexities for the special case where one of the interval bounds is known at indexing time. Finally, those results can be used to efficiently find all pairs of non-overlapping consecutive occurrences.

## 7.1 Top- $k$ Far Consecutive Occurrences

Given a pattern  $P$  and an integer parameter  $k > 0$ , define the *top- $k$  far consecutive occurrences* of  $P$  to be the  $k$  consecutive occurrences of  $P$  in  $S$  with the largest distances. Given a string  $S$  the *string indexing for top- $k$  far consecutive occurrences problem* (SITFCO) is to preprocess  $S$  into a data structure that supports top- $k$  far consecutive occurrences queries. The goal is to obtain a compact data structure while supporting fast queries in terms of the length of the pattern  $P$  and the number of reported occurrences  $k$ .

We can solve the SITFCO problem using the same strategy as for the SITCCO problem, with small modifications. This yields the following result. The details can be found in the full version.

► **Theorem 8.** *Given a string  $S$  of length  $n$  and  $\epsilon$ ,  $0 < \epsilon \leq 1$ , we can build a data structure that can answer top- $k$  far consecutive occurrences queries using either*

- (i)  $O(n \log n)$  space and  $O(m + k)$  query time or
- (ii)  $O(\frac{n}{\epsilon})$  space and  $O(m + k^{1+\epsilon})$  query time.

Here,  $m$  is the length of the query pattern.

## 7.2 Consecutive Occurrences with Gaps

Given a string  $S$  the *string indexing for consecutive occurrences with gaps problem* (SICOG) is to preprocess  $S$  into a compact data structure, such that for any pattern  $P$  and a range  $[\alpha, \beta]$  we can efficiently find all consecutive occurrences of  $P$  where the distance lies within  $[\alpha, \beta]$ . The SICOG problem was considered by Navarro and Thankachan [31] and they give an  $O(n \log n)$  space and  $O(m + \text{occ})$  time solution, where  $\text{occ}$  is the number of consecutive pairs with distance in  $[\alpha, \beta]$ . Using the data structure from Section 3, we get an  $O(n \log n)$  space and  $O(m + \log n + \text{occ})$  time solution for the SICOG problem, which can be optimized using the same strategy as in [31] to achieve the same complexities. However, for a special case of the problem where either  $\alpha$  or  $\beta$  is known at indexing time we can get a similar trade-off as for the SITCCO problem. The following theorem follows from our solution to SITCCO. Again, we refer to the full version for details.

► **Theorem 9.** *Given a string  $S$  of length  $n$  and  $\alpha > 0$ , we can build for any  $\epsilon$  satisfying  $0 < \epsilon \leq 1$  an  $O(\frac{n}{\epsilon})$  space data structure that can answer the following query in  $O(m + \text{occ}^{1+\epsilon})$  time: For a query pattern  $P$  and  $\beta \geq \alpha$ , report all consecutive occurrences of  $P$  in  $S$  where the distance lies in  $[\alpha, \beta]$ . Here,  $m$  is the length of the pattern and  $\text{occ}$  is the number of reported occurrences.*

Similarly, our result for top- $k$  far consecutive occurrences yields the following result for  $\beta$  fixed at indexing time:

► **Theorem 10.** *Given a string  $S$  of length  $n$  and  $\beta > 0$ , we can build for any  $\epsilon$  satisfying  $0 < \epsilon \leq 1$  an  $O(\frac{n}{\epsilon})$  space data structure that can answer the following query in  $O(m + \text{occ}^{1+\epsilon})$  time: For a query pattern  $P$  and  $\alpha$  where  $0 < \alpha \leq \beta$ , report all consecutive occurrences of  $P$  in  $S$  where the distance lies in  $[\alpha, \beta]$ . Here,  $m$  is the length of the pattern and  $\text{occ}$  is the number of reported occurrences.*

### Non-overlapping consecutive occurrences

A natural and well studied variant of string indexing is the problem of finding sets of *non-overlapping* occurrences of a pattern  $P$ . Here, a set of non-overlapping occurrences is a set of occurrences  $\{i_1, \dots, i_k\}$  of  $P$  such that the distance between any two of them is at least  $|P|$ . Several papers study the problem of finding the set of non-overlapping occurrences of maximum size [13, 17, 22, 24]. Note that Theorem 10 applied to  $\alpha = |P|$  solves a different variant of finding sets of non-overlapping occurrences: Namely, finding all pairs of non-overlapping *consecutive* occurrences. We call this problem the *string indexing for non-overlapping consecutive occurrences problem* (SINOCO). The SINOCO problem is inherently different from finding the maximum set of non-overlapping occurrences: For example, the maximum set of non-overlapping occurrences of the pattern  $P = \text{NANA}$  in the string  $S = \text{NANANANA}$  has size 2. However, there are no non-overlapping *consecutive* occurrences. To the best of our knowledge, the SINOCO problem has not been studied before. An immediate corollary of the results in Navarro and Thankachan [31] and Theorem 10 gives the following trade-offs for solving SINOCO:

► **Corollary 11.** *Given a string  $S$  of length  $n$  and  $\epsilon$ ,  $0 < \epsilon \leq 1$ , we can build a data structure that can find all non-overlapping consecutive occurrences of a query pattern  $P$  using either*

- (i)  $O(n \log n)$  space and  $O(m + \text{occ})$  query time or
- (ii)  $O(\frac{n}{\epsilon})$  space and  $O(m + \text{occ}^{1+\epsilon})$  query time.

Here,  $m$  is the length of the query pattern and  $\text{occ}$  is the number of reported occurrences.

**Proof.** Apply the results in [31] and Theorem 10 with  $\beta = n$  and  $\alpha = |P|$ . ◀

## 8 Conclusion and Open Problems

We have introduced the natural problem of string indexing for top- $k$  close consecutive occurrences, and have given both a near-linear space solution achieving optimal query time and a linear space solution achieving a query time that is close to optimal. Using these techniques, we have given new solutions for the problem of string indexing for consecutive occurrences with gaps (SICOG). Furthermore, we have introduced the problem of finding all non-overlapping consecutive occurrences of a pattern (SINOCO) and showed that it can be reduced to a special case of SICOG.

These results open interesting new directions for further research. The most obvious open problem is to see whether it is possible to further improve the results for the main problem considered in this paper, especially, achieve linear space and optimal query time simultaneously. Secondly, it is still open whether it is possible to get an  $O(m + \text{occ})$  time and linear space solution for the special case of the SICOG problem where one of the interval endpoints is fixed, or even  $o(n \log n)$  space for the general problem. For the SINOCO problem, one might find better solutions that do not reduce it to SICOG but use additional insights about the specific structure of the problem. Furthermore, there are many unexplored variations: One could consider indexing for consecutive occurrences of different patterns  $P_1$  and  $P_2$ , chains of consecutive occurrences, together with various distance constraints or top- $k$  queries.

---

### References

- 1 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Minimizing diameters of dynamic trees. In *Proc. 24th ICALP*, pages 270–280, 1997.

## 14:16 String Indexing for Top- $k$ Close Consecutive Occurrences

- 2 Stephen Alstrup, Jacob Holm, and Mikkel Thorup. Maintaining center and median in dynamic trees. In *Proc. 7th SWAT*, pages 46–56, 2000.
- 3 Stephen Alstrup and Theis Rauhe. Improved labeling scheme for ancestor queries. In *Proc. 13th SODA*, pages 947–953, 2002.
- 4 Johannes Bader, Simon Gog, and Matthias Petri. Practical variable length gap pattern matching. In *Proc. 15th SEA*, pages 1–16, 2016.
- 5 Philip Bille and Inge Li Gørtz. The tree inclusion problem: In linear space and faster. *ACM Trans. Algorithms*, 7(3):1–47, 2011.
- 6 Philip Bille and Inge Li Gørtz. Substring range reporting. *Algorithmica*, 69(2):384–396, 2014.
- 7 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory Comput. Syst.*, 55(1):41–60, 2014.
- 8 Sudip Biswas, Arnab Ganguly, Rahul Shah, and Sharma V Thankachan. Ranked document retrieval for multiple patterns. *Theor. Comput. Sci.*, 746:98–111, 2018.
- 9 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.
- 10 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online sorted range reporting. In *Proc. 30th ISAAC*, pages 173–182, 2009.
- 11 Manuel Cáceres, Simon J Puglisi, and Bella Zhukova. Fast indexes for gapped pattern matching. In *Proc. 46th SOFSEM*, pages 493–504, 2020.
- 12 Timothy M Chan. Persistent predecessor search and orthogonal point location on the word ram. *ACM Trans. Algorithms*, 9(3):1–22, 2013.
- 13 Hagai Cohen and Ely Porat. Range non-overlapping indexing. In *Proc. 20th ISAAC*, pages 1044–1053, 2009.
- 14 James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.
- 15 Greg N Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and  $k$  smallest spanning trees. *SIAM J. Comput.*, 26(2):484–538, 1997.
- 16 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with  $O(1)$  worst case access time. *J. ACM*, 31(3):538–544, 1984.
- 17 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Succinct non-overlapping indexing. *Algorithmica*, 82(1):107–117, 2020.
- 18 Wing-Kai Hon, Manish Patil, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Indexes for document retrieval with relevance. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 351–362, 2013.
- 19 Wing-Kai Hon, Manish Patil, Rahul Shah, and Shih-Bin Wu. Efficient index for retrieving top- $k$  most frequent documents. *J. Discrete Algorithms*, 8(4):402–417, 2010.
- 20 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top- $k$  string retrieval. *J. ACM*, 61(2):1–36, 2014. Announced at 50th FOCS.
- 21 Wing-Kai Hon, Sharma V. Thankachan, Rahul Shah, and Jeffrey Scott Vitter. Faster compressed top- $k$  document retrieval. In *Proc. 23rd DCC*, pages 341–350, 2013.
- 22 Sahar Hooshmand, Paniz Abedin, M. Oguzhan Külekci, and Sharma V. Thankachan. Non-overlapping indexing - cache obliviously. In *Proc. 29th CPM*, pages 8:1–8:9, 2018.
- 23 Costas S Iliopoulos and M Sohel Rahman. Indexing factors with gaps. *Algorithmica*, 55(1):60–70, 2009.
- 24 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. Range non-overlapping indexing and successive list indexing. In *Proc. 11th WADS*, pages 625–636, 2007.
- 25 Moshe Lewenstein. Indexing with gaps. In *Proc. 18th SPIRE*, pages 135–143, 2011.
- 26 J. Ian Munro, Gonzalo Navarro, Jesper Sindahl Nielsen, Rahul Shah, and Sharma V. Thankachan. Top- $k$  term-proximity in succinct space. *Algorithmica*, 78(2):379–393, 2017. Announced at 25th ISAAC.

- 27 J. Ian Munro, Gonzalo Navarro, Rahul Shah, and Sharma V. Thankachan. Ranked document selection. *Theor. Comput. Sci.*, 812:149–159, 2020.
- 28 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):1–47, 2014.
- 29 Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comput.*, 46(1):80–113, 2017. Announced at 23rd SODA.
- 30 Gonzalo Navarro and Sharma V. Thankachan. New space/time tradeoffs for top-k document retrieval on sequences. *Theor. Comput. Sci.*, 542:83–97, 2014. Announced at 20th SPIRE.
- 31 Gonzalo Navarro and Sharma V. Thankachan. Reporting consecutive substring occurrences under bounded gap constraints. In *Proc. 26th CPM*, pages 367–373, 2015.
- 32 Rahul Shah, Cheng Sheng, Sharma V. Thankachan, and Jeffrey Scott Vitter. Top-k document retrieval in external memory. In *Proc. 21st ESA*, pages 803–814, 2013.
- 33 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.
- 34 Dekel Tsur. Top-k document retrieval in optimal space. *Inf. Process. Lett.*, 113(12):440–443, 2013.
- 35 Peter Weiner. Linear pattern matching algorithms. In *Proc. 14th FOCS*, pages 1–11, 1973.



# Fair Tree Connection Games with Topology-Dependent Edge Cost

**Davide Bilò** 

Department of Humanities and Social Sciences, University of Sassari,  
Via Roma 151, 07100 Sassari (SS), Italy  
davide.bilo@uniss.it

**Tobias Friedrich** 

Hasso Plattner Institute, University of Potsdam,  
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany  
tobias.friedrich@hpi.de

**Pascal Lenzner** 

Hasso Plattner Institute, University of Potsdam,  
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany  
pascal.lenzner@hpi.de

**Anna Melnichenko** 

Hasso Plattner Institute, University of Potsdam,  
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany  
anna.melnichenko@hpi.de

**Louise Molitor** 

Hasso Plattner Institute, University of Potsdam,  
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany  
louise.molitor@hpi.de

---

## Abstract

How do rational agents self-organize when trying to connect to a common target? We study this question with a simple tree formation game which is related to the well-known fair single-source connection game by Anshelevich et al. (FOCS'04) and selfish spanning tree games by Gourvès and Monnot (WINE'08). In our game agents correspond to nodes in a network that activate a single outgoing edge to connect to the common target node (possibly via other nodes). Agents pay for their path to the common target, and edge costs are shared fairly among all agents using an edge. The main novelty of our model is dynamic edge costs that depend on the in-degree of the respective endpoint. This reflects that connecting to popular nodes that have increased internal coordination costs is more expensive since they can charge higher prices for their routing service.

In contrast to related models, we show that equilibria are not guaranteed to exist, but we prove the existence for infinitely many numbers of agents. Moreover, we analyze the structure of equilibrium trees and employ these insights to prove a constant upper bound on the Price of Anarchy as well as non-trivial lower bounds on both the Price of Anarchy and the Price of Stability. We also show that in comparison with the social optimum tree the overall cost of an equilibrium tree is more fairly shared among the agents. Thus, we prove that self-organization of rational agents yields on average only slightly higher cost per agent compared to the centralized optimum, and at the same time, it induces a more fair cost distribution. Moreover, equilibrium trees achieve a beneficial trade-off between a low height and low maximum degree, and hence these trees might be of independent interest from a combinatorics point-of-view. We conclude with a discussion of promising extensions of our model.

**2012 ACM Subject Classification** Theory of computation → Algorithmic game theory; Theory of computation → Quality of equilibria; Theory of computation → Convergence and learning in games; Theory of computation → Network formation

**Keywords and phrases** Network Design Games, Spanning Tree Games, Fair Cost Sharing, Price of Anarchy, Nash Equilibrium, Algorithmic Game Theory, Combinatorics



© Davide Bilò, Tobias Friedrich, Pascal Lenzner, Anna Melnichenko, and Louise Molitor;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 15; pp. 15:1–15:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.15

Related Version A full version of the paper is available at [9], <http://arxiv.org/abs/2009.10988>.

**Acknowledgements** We thank Warut Suksompong for many interesting discussions. Moreover, we are grateful to our anonymous reviewers for their valuable suggestions. This work has been partly supported by COST Action CA16228 European Network for Game Theory (GAMENET).

## 1 Introduction

Network Design is an important optimization problem where for a given weighted host graph and a given set of terminal pairs the cheapest subgraph which connects all terminal pairs has to be found. Besides an abundance of research works with an optimization point-of-view, e.g. see the survey by Magnanti and Wong [28], a strategic version of the Network Design problem [5, 4] has kindled significant interest in recent years. In the *connection game*, a weighted host graph  $H$  is given and  $n$  agents with given terminal node pairs  $(s_i, t_i)$ , for  $1 \leq i \leq n$ , strategically select  $s_i$ - $t_i$ -paths in  $H$  to connect their respective terminal nodes. The union of the selected paths forms a subgraph  $G$  of  $H$  which constitutes the actually designed network. The usage cost of each edge of  $H$  corresponds to its weight, and agents using some edge  $e$  in  $H$  have to pay this cost. If an edge  $e$  is used by more than one agent, then a cost-sharing protocol determines how the usage cost of  $e$  is split among its users. One of the most common cost-sharing protocols is Shapley cost-sharing where each agent pays a fair share of the edge cost, i.e., the cost-share is the edge cost divided by the number of users. This game-theoretic setting, called *fair connection game*, was investigated by Anshelevich et al. [4] and has since become an influential paper in Algorithmic Game Theory. An important special case is the setting in which all the strategic agents want to connect to a common source node. This variant, where  $t_1 = \dots = t_n$  and where every other node is a terminal node of some agent, is usually denoted as the (*fair*) *single-source connection game*, with the interpretation that all the agents want to connect to a common source node to receive broadcast messages and that the edge cost for connecting to the common source is paid by the downstream users.

A similar related game-theoretic setting are *selfish spanning tree games* [21]. There a weighted complete host graph with  $n + 1$  nodes, consisting of a common target node  $r$  and  $n$  nodes which correspond to selfish agents, is given and every selfish agents now selects an incident edge to connect to the common target node  $r$  either directly or indirectly via selected edges of other agents. The cost of an agent is then determined by its unique path to  $r$ . Thus, in any equilibrium the subgraph of all selected edges forms a spanning tree rooted at  $r$ .

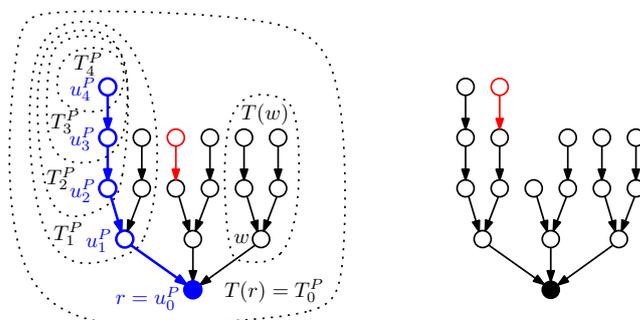
This paper sets out to investigate a game-theoretic Network Design model that is closely related to the fair single-source connection game and to selfish spanning tree games. The main novel feature of our model is the twist that the cost of the edges in the formed spanning tree depend on its topology. In particular, we consider dynamic edge costs which are proportional to the in-degree of the node they connect to. Network nodes with high in-degree can be considered as popular, and we assume that connecting to popular nodes is more expensive than connecting to unpopular nodes. These dynamic edge costs can also be understood as the internal cost of a node for coordinating data traffic coming from different connections. A node with many incoming edges and thus higher internal coordination cost can charge higher prices for serving each of the incoming edges.

To the best of our knowledge, we define and analyze the first (game-theoretic) Network Design model where the edge costs depend on the topology of the formed network. We believe that this model sheds light on settings where the actual charges for establishing links are determined by supply and demand and the agents act strategically to optimize their cost for receiving their desired service.

## 1.1 Model, Definition, Notation

We consider a strategic game called *fair tree connection game with topology-dependent edge cost*, or *tree connection game (TCG)* for short. In the TCG we will consider a given unweighted complete directed host graph  $H = (V, E)$ , where  $V$  is the set of nodes and  $E$  is the set of edges of  $H$ . The host graph  $H$  consists of  $n + 1$  nodes  $V = \{r, v_1, \dots, v_n\}$  where node  $r$  is the common target node, also called the root, and every node  $v_i$ , for  $1 \leq i \leq n$ , corresponds to a selfish agent  $i$  striving to be connected to the root  $r$ . For this, every agent  $i$  strategically activates a single incident edge  $(v_i, s_i)$ , where  $s_i \in V \setminus \{v_i\}$ . Hence, the strategy space of each agent is the set of other nodes to connect to. Given a strategy profile  $\mathbf{s} = (s_1, \dots, s_n)$ , i.e., an  $n$ -dimensional vector where the  $j$ -th entry corresponds to the node to which agent  $j$  wants to activate her edge, we consider the directed network  $T(\mathbf{s}) = (V, E(\mathbf{s}))$  which is induced by all the activated edges, i.e.,  $E(\mathbf{s}) = \{(v_i, s_i) \mid 1 \leq i \leq n\}$ . We will see later that  $T(\mathbf{s})$  is a spanning tree rooted at  $r$  if  $\mathbf{s}$  is an equilibrium state of the TCG, hence the name.

The cost of agent  $i$  in the network  $T(\mathbf{s})$  depends on its unique path  $P_i$  in  $T(\mathbf{s})$  to the root  $r$  (if such a path exists). In case of existence, the path  $P_i$  must be unique, since the out-degree of every node in  $T(\mathbf{s})$  is at most 1. More precisely, let  $P_i$  be the directed path from  $v_i$  to  $r$  in  $T(\mathbf{s})$ , let  $\text{indeg}_{T(\mathbf{s})}(v)$  denote the number of edges with endpoint  $v$  in  $T(\mathbf{s})$ , let  $T(u)$  denote the subgraph of  $T(\mathbf{s})$  rooted at node  $u$ , i.e., the subgraph of  $T(\mathbf{s})$  induced by the nodes  $u$  and every node which has a directed path to  $u$  and let  $|T(u)|$  denote the number of nodes in  $T(u)$ . See Figure 1.



■ **Figure 1** Left:  $T(\mathbf{s})$  for  $n = 16$  agents. The path  $P$  is colored blue and we have  $d_0^P = 3, d_1^P = 2, d_2^P = d_3^P = 1, d_4^P = 0$  and  $|T_1^P| = 6, |T_2^P| = 3, |T_3^P| = 2, |T_4^P| = 1, |T(w)| = 5$ . Nodes  $w$  and  $u_1^P$  and also their corresponding agents are siblings. The shown network  $T(\mathbf{s})$  is not stable since the agent colored red with cost  $1 + \frac{2}{2} + \frac{3}{5} = \frac{13}{5}$  has an improving move. Right: the network after the agent colored red improved its cost to  $1 + \frac{1}{2} + \frac{2}{3} + \frac{3}{7} = \frac{109}{42} < \frac{13}{5}$ .

The cost of agent  $i$  in  $T(\mathbf{s})$  then is  $\text{cost}_{T(\mathbf{s})}(i) := \sum_{(u,v) \in P_i} \frac{\text{indeg}_{T(\mathbf{s})}(v)}{|T(u)|}$ , if  $P_i$  exists and  $\infty$  otherwise. This cost function has the following very natural interpretation: the cost of activating edge  $(u, v)$  from node  $u$  to node  $v$  is equal to node  $v$ 's in-degree, and this cost is fairly shared by all agents who use edge  $(u, v)$  on their path towards the root  $r$ , i.e., by all agents in  $T(u)$ . We assume that each agent activates a single edge strategically to minimize its cost in the induced network  $T(\mathbf{s})$ . Clearly, since every agent  $i$  can activate the edge  $(v_i, r)$ ,  $i$  can enforce finite cost by enforcing that the path  $P_i$  exists.

Consider a strategy profile  $\mathbf{s} = (s_1, \dots, s_{i-1}, s_i, s_{i+1}, \dots, s_n)$ . We say that agent  $i$  has an *improving move* in  $\mathbf{s}$  if  $i$  has some alternative strategy  $s'_i \neq s_i$  such that for the induced strategy profile  $\mathbf{s}' = (s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_n)$  we have  $cost_{T(\mathbf{s}')} (i) < cost_{T(\mathbf{s})} (i)$ , i.e., agent  $i$  can strictly decrease its cost by activating a different outgoing edge. With this, we define the strategy profile  $\mathbf{s}$  to be in *pure Nash equilibrium (NE)* or to be *stable* if no agent has an improving move in  $\mathbf{s}$ . If the context is clear, we use strategy profiles and their induced network interchangeably, i.e., we say that the network  $T(\mathbf{s})$  is in NE or stable, if  $\mathbf{s}$  is in NE. Moreover, when we refer to some network  $T(\mathbf{s})$  we will from now on omit the reference to the strategy profile  $\mathbf{s}$  and call the network simply  $T$ . Every stable network  $T$  must be a spanning tree rooted at  $r$ , since every agent  $i$  can activate the edge  $(v_i, r)$  to achieve finite cost.

The *social cost*  $SC(T)$  of a network  $T$  is simply the sum over all agents' costs, i.e.,

$$SC(T) = \sum_{i=1}^n cost_T(i) = \sum_{v_i \in V} \sum_{(u,v) \in P_i} \frac{indeg_T(v)}{|T(u)|} = \sum_{(u,v) \in E} \frac{indeg(v)}{|T(u)|} \cdot |T(u)| = \sum_{v \in V} (indeg(v))^2.$$

Note that  $SC(T)$  nicely reflects the overall cost impact of the nodes' popularity or coordination costs which scales quadratically with the in-degree of a node. For a given number of agents  $n$ , let  $OPT_n$  denote the network which minimizes the social cost. Moreover, if stable networks exist for  $n$  agents, we let  $worstNE_n$  denote the stable network with the highest social cost and  $bestNE_n$  the stable network with the lowest social cost. We define the *Price of Anarchy (PoA)* [27] as  $PoA = \sup_n \frac{SC(worstNE_n)}{OPT_n}$  and the *Price of Stability (PoS)* [4] as  $PoS = \sup_n \frac{SC(bestNE_n)}{OPT_n}$ , where the supremum is taken over all  $n$  that admit a stable network. Besides the PoA and the PoS, that both focus on the overall cost and compare with the cost of a centrally designed social optimum network, we use a measure of the quality of networks which focuses on the cost distribution among the agents, called the *Fairness Ratio (FR)*, analogously to the *utility uniformity* introduced in [19]. For a given network  $T$ , the  $FR(T)$  is the ratio between the maximum and the minimum cost incurred by any agent, i.e.,  $FR(T) := \frac{\max_{v_i \in V} cost_T(i)}{\min_{v_i \in V} cost_T(i)}$ .

Finally, we introduce some additional notation for arguing about the designed networks  $T(\mathbf{s})$ . (See Fig. 1 for an illustration). For our analysis we use directed paths in  $T(\mathbf{s})$  which start at some non-root node  $z$  and end at the root  $r$ . Let  $P$  be such a path of length  $\ell \in \mathbb{N}$ . We denote by  $u_j^P$  the node on  $P$  which is at distance  $j$  to  $r$ , hence the root  $r$  is denoted by  $u_0^P$  and node  $z$  by  $u_\ell^P$ . Moreover, let  $T_j^P := T(u_j^P)$  and we use  $d_j^P$  for the in-degree of a node with distance  $j$  from the root  $r$  on path  $P$ , hence,  $d_j^P := indeg_{T(\mathbf{s})}(u_j^P)$ . We omit the reference to path  $P$  whenever it is clear from the context.

## 1.2 Related Work

Our model is closely related to several models that have been intensively studied.

We start with the (fair) single-source connection game [5, 4] which we already briefly discussed in the introduction. The key feature of this game is that agents strategically select a set of edges to connect their respective terminals. The cost of each edge is shared among all the agents who selected the respected edge. While in [5] and later also in [24] arbitrary cost sharing is considered, the paper [4] focuses on fair cost sharing which can be derived from the Shapley value [33]. For this Anshelevich et al. [4] show that stable networks always exist since the game is a potential game [32], additionally they prove that the PoA is  $n$  and the PoS is upper bounded by  $H_n$ , where  $H_n$  is the  $n$ -th harmonic number. For a given directed host graph this bound on the PoS is tight but the case for undirected host networks is still a major open problem. More is known for single-source connection games on undirected

networks. Chekuri et al. [12] show that the PoA is in  $\mathcal{O}(\sqrt{n} \log^2 n)$  if the agents join the game sequentially and play their respective best response. A PoS in  $\mathcal{O}(\log \log n)$  was proven by Fiat et al. [20] for the special case where all nodes of the given network correspond to a terminal of some agent. Finally, Bilò et al. [8] prove a constant PoS for the fair single-source connection game on undirected networks. Moreover, Albers and Lenzner [1] show that the optimum is a  $H_n$ -approximate Nash equilibrium for the fair single-source connection game. In contrast to our model, the cost of an edge in the (fair single-source) connection game is given via a positively weighted host network. Hoefer and Krysta [25] investigate a variant with edge weights derived from a geometry.

Also selfish spanning tree games [21] are close to our model and we already briefly discussed them in the introduction. The key difference to our model is that a weighted complete network is given and that the cost of an agent is defined differently. Gourvès and Monnot [21] define three variants of the agents' cost function: either it is the weight of the first edge on the path to the common root  $r$ , or the minimum or maximum weight edge on the entire path towards  $r$ . Cost sharing is not considered. The authors prove bounds on the PoA which vary from unbounded to 1 depending on the exact setting. The games in [21] are inspired by the classical problem of allocating the cost of a spanning tree among its nodes by Claus and Kleitman [13] and its variant from cooperative game theory considered by Bird [10]. Later, Granot and Huberman [22, 23] considered minimum cost spanning tree games and different cost allocation protocols for this have been considered by Escoffier et al. [17]. The key difference of all these models to our model is that a cooperative game is considered which is a stark contrast to our non-cooperative setting. Also game-theoretic topology control problems are related to spanning tree games and our model. Eidenbenz et al. [16] consider a setting where a set of agents which correspond to wireless devices want to connect terminal nodes, whereas Mittal et al. [31] consider wireless access point selection by selfish agents.

Also classical network formation games [26, 6, 18] are related to our model. There the agents correspond to nodes in a network and every agent buys a set of incident edges to connect to other agents. The goal of each agent is to create a connected network and to occupy a central position in this network. For the influential network creation game of Fabrikant et al. [18], that has a parameter  $\alpha$  for the trade-off between edge cost and distance costs, the PoA was shown to be constant for almost all values of  $\alpha$  [14, 2]. For high values of  $\alpha$  all equilibrium networks of these games are known to be trees [30, 29, 7]. A variant of the network creation game where agents can only buy a single edge was considered by Ehsani et al. [15]. Most notably, the topology dependent edge costs that we employ in our model were proposed by Chauhan et al. [11] for the network creation game [18]. To the best of our knowledge, this is the only setting where topology dependent edge costs have been considered.

### 1.3 Our Contribution

We study a novel game-theoretic model for the formation of a tree network which is related to the well-known fair single-source connection game by Anshelevich et al. [4, 5] and to selfish spanning tree games by Gourvès and Monnot [21]. The key difference of our model is that we consider dynamic edge costs which depend on the topology of the created spanning tree. In particular, the cost of an edge is equal to the in-degree of its endpoint. This specific choice was proposed in [11] for the classical network creation game [18] and we transfer this idea to the Network Design domain. Our analysis holds for any edge cost function of the form  $\alpha$  times the in-degree of the target node, for any constant  $\alpha$ . However, our general approach is valid also for edge cost functions that depend non-linearly on the degrees of the involved nodes.

Regarding the existence of stable trees we show that our model is in stark contrast to the models in [4, 18, 21] since in our model stable trees may not exist. In particular, we show that our game has no NE for  $n = 16$  and  $n = 18$  which implies that the TCG cannot admit a potential function. This is contrasted with a proof that for infinitely many  $n$  stable trees do exist, and we conjecture that we have found all examples for NE non-existence. Towards investigating the quality of the equilibrium networks of our model, we first provide a rigorous study of the structural properties of stable trees. We show that every stable tree consists of stable subtrees and that the height of any stable tree is in  $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ . For the root  $r$ , which turns out to be the node with the highest in-degree in any stable network, we show that its in-degree is between  $\Omega\left(\frac{\log n}{\log \log n}\right)$  and  $2^{\mathcal{O}(\sqrt{\log n})}$ . This shows that the maximum internal coordination overhead of a single node in any stable tree is rather small.

Our main results are on the quality of equilibrium trees. By using the established structural properties and a connection to the Riemann zeta function we obtain an upper bound on the PoA of 8.62 which is contrasted with a lower bound of 2.4317. For the PoS we derive a lower bound of  $\frac{7}{5} - \varepsilon$ . Moreover, we give for an infinite number of values for  $n$  an upper bound of 2.83 on the PoS. Regarding the Fairness Ratio, we first show that the socially optimal tree is rather unfair, i.e., having a Fairness Ratio of  $n \cdot H_n$ . In contrast, we prove that any equilibrium tree has a Fairness Ratio in  $o(n)$ .

This shows that stable trees have only slightly higher social cost compared to the social optimum. In particular, on average every agent pays only a constant factor more than the trivial lower bound for any spanning tree. At the same time stable trees are more fair, have low height and low in-degrees.

We conclude with a brief discussion of the path version extension of our model, where agents select paths as strategies as in [5, 4]. This extension seems promising for future work since we show that allowing a richer strategy space yields a larger set of equilibria and we give equilibria for  $n = 16$  and  $n = 18$ . Hence, in the path-version equilibria may always exist, but the PoA could be higher.

We refer to [9] for all details which were omitted due to space constraints.

## 2 Structure and Properties of Equilibrium Trees

It is clear that each agent can compute her best response in polynomial time as the number of possible strategies for an agent is  $n$ , and the agent can easily compute her cost in linear time. In the following we show that any stable tree consists of stable subtrees, we prove an upper bound of  $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$  to the number of edges of any leaf-to-root path of any stable network, and in the end, we provide bounds on the degree of the root. We start with the statement that any stable tree consists of stable subtrees.

► **Lemma 1.** *If  $T$  is stable, then any subtree  $T(x)$  is stable in the corresponding subgame.*

Next, we will consider the height of a stable network and need the following technical lemmas.

► **Lemma 2.** *Let  $k \in \mathbb{N}$  be the length of a fixed leaf-to-root-path  $P$  in a stable network  $T$ . Then, for every  $1 < i < k$ ,  $d_{i-1} \geq \frac{|T_i|}{|T_i| - |T_{i+1}|}(d_i - 1)$ .*

Since  $|T_{i+1}| > 0$ , Lemma 2 yields that the sequence  $d_0, d_1, \dots, d_k$ , is monotonically decreasing.

► **Corollary 3.** *Let  $k \in \mathbb{N}$  be the length of a leaf-to-root-path  $P$  in a stable network  $T$ . Then, for every  $1 < i < k$ ,  $d_i \geq d_{i+1}$ .*

The next lemma shows that the in-degree of nodes strictly decreases after a constant number of hops.

► **Lemma 4.** *Let  $k \in \mathbb{N}$  be the length of a fixed leaf-to-root-path  $P$  in a stable network  $T$ . Then, for every subtree  $T(v)$  with  $|T(v)| > 4$  and for every  $1 < i < k - 2$  we have  $d_{i-1} > d_{i+1}$ .*

In the following we investigate upper and lower bounds on the in-degree of the root in stable trees. More precisely, we show an upper bound of  $2^{O(\sqrt{\log n})}$  and a lower bound of  $\Omega(\log n / \log \log n)$ .

► **Theorem 5.** *The in-degree of the root in a stable network  $T$  is at least  $\frac{\ln(4\sqrt{n/5})}{\ln \ln(4\sqrt{n/5})}$ .*

To give an upper bound on the in-degree of the root, we first have to provide the following technical lemmas. The first technical lemma bounds the in-degree of the parent of any leaf.

► **Lemma 6.** *In a stable network  $T$  the in-degree of the parent of any leaf is 1.*

The second technical lemma shows how the in-degrees of two sibling nodes are related.

► **Lemma 7.** *Consider a subtree  $T(x)$  of a stable network  $T$ . Then  $\text{indeg}(x) \leq \text{indeg}(v) \cdot \left(1 + \frac{|T(u)|}{|T(v)|}\right) + 1$ , where  $v$  and  $u$  are different children of  $x$ .*

From Lemma 7, we derive the following remark and corollary.

► **Remark 8.** Consider a subtree  $T(x)$  in a stable network  $T$ . Then  $\text{indeg}(x) \leq 2 \cdot \text{indeg}(v) + 1$ , where  $v$  is a root of the second smallest subtree of  $T(x)$ .

► **Corollary 9.** *If  $T$  is a stable network, then every node  $u$  in  $T$  has at least  $\text{indeg}(u) - 1$  children of in-degree at least  $(\text{indeg}(u) - 1)/2$ .*

Now we can prove an upper bound to the in-degree of the root of any stable tree.

► **Theorem 10.** *The in-degree of the root in a stable network  $T$  is  $2^{O(\sqrt{\log n})}$ .*

**Proof.** Let  $T$  be a stable tree of height  $h$ . Let  $v_h, \dots, v_0$  be a leaf-to-root path. Note that the in-degree of the root  $v_0$  is maximal if the in-degree of each node in the  $v_h$ - $v_0$ -path is maximal, i.e., by Lemma 7 and 6, it corresponds to the in-degree sequence  $D := (0, 1, d_{h-2}, \dots, d_0)$ , where  $d_{i-1} = 2d_i + 1$ .

Next, we show that nodes at distance  $h - 2$  from the root can have an in-degree of at most 2. Assume to the contrary that there is a node  $u$  having an edge to a node  $x$  such that  $\text{indeg}(x) = 3$  and  $x$  is at distance  $h - 2$  from the root  $v_0$ . As we have proved above, the in-degree of all children of  $x$  is at most 1. Thus,  $u$  can swap to any leaf node of the subtree  $T(x)$ . Let  $T'$  be the tree obtained after  $u$  swapped. If  $u$  swaps to a child of  $x$ , it decreases its cost by  $\text{cost}_T(u) - \text{cost}_{T'}(u) = \frac{3}{2} - \frac{1}{2} - \frac{2}{3} > 0$ , i.e., it is an improving move. The swap to a leaf node at distance 2 from  $x$  implies an improvement by  $\text{cost}_T(u) - \text{cost}_{T'}(u) = \frac{3}{2} - \frac{1}{2} - \frac{1}{3} - \frac{2}{4} > 0$ , i.e., it is an improvement. Since  $T$  is stable, we get a contradiction. Thus,  $D = (0, 1, 2, 5, 11, \dots, d_0)$ , i.e.,

$$d_i = 3 \cdot 2^{h-i-2} - 1 \text{ for } i \leq h - 3, \text{ where } d_h = 0, d_{h-1} = 1, d_{h-2} = 2. \quad (1)$$

We now estimate the minimum possible number of nodes in the tree  $T$ . By Corollary 9 if the in-degree of a node is equal to  $k$ , then it has at least  $k - 1$  children with an in-degree of at least  $(k - 1)/2$ . Thus, starting from the root, the in-degrees of the nodes on each level decrease no more than twice. Hence, the total size of the tree is at least

$$\sum_{i=1}^{h-1} \left( d_i \prod_{j=0}^{i-1} (d_j - 1) \right) > \sum_{i=1}^{h-1} \prod_{j=0}^{i-1} 2^{h-j-2} > 2^{\sum_{j=0}^{h-3} (h-j-2)} = 2^{\frac{(h-1)(h-2)}{2}},$$

where  $h$  is the height of  $T$ . Thus,  $h < \frac{3 + \sqrt{1 + 8 \log n}}{2}$ . With equation (1), this implies  $d_0 \in 2^{O(\sqrt{\log n})}$ .  $\blacktriangleleft$

Now we are able to show that the length of any node-to-root path is  $O\left(\frac{\log n}{\log \log n}\right)$ .

► **Theorem 11.** *If  $T$  is a stable network, then its height  $h \in O\left(\frac{\log n}{\log \log n}\right)$ .*

**Proof.** Consider a leaf-to-root path  $P$  in  $T$ . We show that there are  $O\left(\frac{\log n}{\log \log n}\right)$  indices such that  $|T_i^P| - |T_{i+1}^P| \geq \sqrt[3]{\log n} \cdot |T_{i+1}^P|$  and  $O\left(\frac{\log n}{\log \log n}\right)$  indices such that  $|T_i^P| - |T_{i+1}^P| < \sqrt[3]{\log n} \cdot |T_{i+1}^P|$ .

Let  $k$  be the number of indices  $i$  that satisfy  $|T_i^P| - |T_{i+1}^P| \geq \sqrt[3]{\log n} \cdot |T_{i+1}^P|$ . Then we have that  $|T_i^P| = |T_i^P| - |T_{i+1}^P| + |T_{i+1}^P| > \sqrt[3]{\log n} \cdot |T_{i+1}^P|$ . Since  $|T_i^P| > |T_{i+1}^P|$  for every  $i$ , and because  $|T_i^P| \leq n$ , we have that  $|T_0^P| > (\log n)^{k/3}$  and  $|T_0^P| = n + 1$ , from which we derive  $(\log n)^{k/3} \leq n$ , i.e.,  $k = O\left(\frac{\log n}{\log \log n}\right)$ .

By Lemma 4 and Corollary 3, there are  $O(\sqrt[3]{\log n}) = O\left(\frac{\log n}{\log \log n}\right)$  indices  $i$  such that  $d_i^P \leq 4\sqrt[3]{\log n}$ . We now prove that there are  $O\left(\frac{\log n}{\log \log n}\right)$  indices such that  $|T_i^P| - |T_{i+1}^P| < \sqrt[3]{\log n} \cdot |T_{i+1}^P|$  and such that  $d_i^P \geq 4\sqrt[3]{\log n}$ . By Lemma 2 and using the fact that  $d_i^P \geq 4\sqrt[3]{\log n}$  and  $n \geq 2$ , we have that

$$d_{i-1}^P \geq \frac{|T_i^P|}{|T_i^P| - |T_{i+1}^P|} (d_i^P - 1) \geq \frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}} (d_i^P - 1) \geq \sqrt{\frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}} d_i^P.$$

By Corollary 3 we have that  $d_{i-1}^P \geq d_i^P$  for every  $i$ . Hence  $d_0 \geq \left(\frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}\right)^{k/2}$ . Moreover, by Theorem 10,  $d_0^P \leq 2^\alpha \sqrt{\log n}$  for some constant  $\alpha > 0$ . As a consequence, we have that  $\left(\frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}\right)^{k/2} \leq 2^\alpha \sqrt{\log n}$ , i.e.,  $2^{\frac{k}{2} \log \frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}} \leq 2^\alpha \sqrt{\log n}$ , which implies,  $k \leq 2\alpha \frac{\sqrt{\log n}}{\log(1 + \sqrt[3]{\log n})}$ .

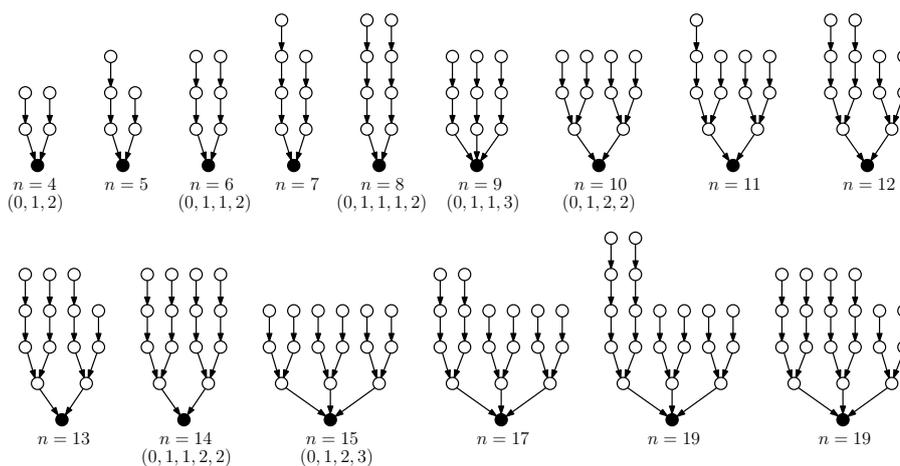
We complete the proof by showing that  $\frac{\sqrt{\log n}}{\log \frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}} \leq 3 \frac{\log n}{\log \log n}$ , i.e., we have to show that  $\log \log n \leq 3\sqrt{\log n} \cdot \log \frac{1 + \sqrt[3]{\log n}}{\sqrt[3]{\log n}}$ . Let  $M = \sqrt[3]{\log n}$ . We have to prove that

$$\log M \leq M^{3/2} \log \frac{1 + M}{M} = \log \left( \frac{1 + M}{M} \right)^{M^{3/2}}. \quad (2)$$

By Bernoulli's inequality,  $(1 + \frac{1}{M})^{M^{3/2}} \geq 2^{M^{1/2}} \geq M$  for  $M \geq 16$ . Thus, inequality (2) is satisfied.  $\blacktriangleleft$

### 3 Existence of Equilibrium Trees

In this section we analyze whether the TCG admits equilibrium trees for all agent numbers  $n$ . We first show that in general equilibrium existence is not guaranteed since for  $n = 16$  and  $n = 18$  no stable tree exists. We contrast this negative result with a NE existence proof for infinitely many agent numbers  $n$ . This positive result is achieved for so-called balanced trees, i.e., trees where all nodes with the same distance to the root have the same in-degree. We believe that our positive results can be strengthened to proving that stable trees exist for all  $n$  except  $n = 16$  and  $n = 18$ , and we leave this as an intriguing open problem. Figure 2 shows sample equilibrium trees for small  $n$ .



■ **Figure 2** Sample equilibrium trees for  $n = 4$  to  $n = 19$ . All depicted trees for  $n < 19$  are the unique equilibria for the respective  $n$ . For  $n = 19$  two equilibrium trees exist. No stable tree exists for  $n = 16$  and  $n = 18$ . The stable trees for  $n = 4, 6, 8, 9, 10, 14, 15$  are balanced trees and are annotated with their identifying in-degree sequence of all leaf-to-root paths. (See Section 3.1 for definitions.)

► **Theorem 12.** *For  $n = 16$  there exists no stable network.*

The non-existence of a stable tree for  $n = 16$  directly implies that the TCG cannot have the finite improvement property, which states that every sequence of improving moves must be finite, i.e., reaches a Nash equilibrium. Thus, since the finite improvement property is equivalent to the game admitting a potential function [32] this implies the following statement.

► **Corollary 13.** *The TCG is not a potential game.*

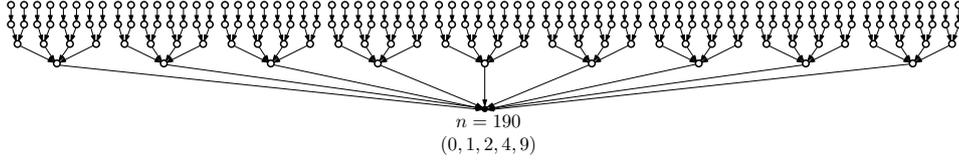
► **Remark 14.** By computational experiments we have obtained equilibrium trees for the TCG for  $1 \leq n \leq 100$ , except for  $n = 16$  and  $n = 18$ . For  $n = 18$  we have verified via a brute-force search over all possible trees that no stable tree exists. Interestingly, for  $n \geq 19$  equilibrium trees are no longer unique and in general the number of non-isomorphic equilibrium trees grows as  $n$  grows.

#### 3.1 Balanced Trees

Despite the negative result of the non-existence of a stable tree for  $n = 16$ , in this section we prove the existence of NE's for infinitely many values of  $n$ . We prove this result by showing an interesting set of conditions for ruling out potential edge swaps; the proved conditions

## 15:10 Fair Tree Connection Games with Topology-Dependent Edge Cost

altogether allow us to show that there are infinitely many (balanced) trees that are stable. More precisely, we say that  $T$  is *balanced* if any two nodes at the same distance from the root  $r$  have equal in-degrees. Note, that any balanced tree  $T$  of height  $h$  can be uniquely encoded by a sequence of node degrees  $(0, d_{h-1}, \dots, d_0)$ , where  $d_i$  is an in-degree of nodes at level  $i$ , i.e., at distance  $i$  from the root. In this section we show that all the balanced trees of the form  $(0, 1, 2, 4, d_{h-4}, \dots, d_0)$  such that  $d_i < d_{i-1} \leq 2d_i + 1$ , for every  $1 \leq i \leq h - 4$  are stable. (See Figure 3 for an example.)



■ **Figure 3** Sample of an extremal balanced tree with degree sequence  $(0, 1, 2, 4, 9)$ .

► **Theorem 15.** *The balanced tree  $T$  with degree sequence  $(0, 1, 2, 4, d_{h-4}, \dots, d_0)$ , where  $d_{j+1} < d_j \leq 2d_{j+1} + 1$  for every  $j \leq h - 4$ , is stable.*

From Theorem 15 we derive the following corollary.

► **Corollary 16.** *The TCG with  $n$  agents admits a NE for infinitely many values of  $n \in \mathbb{N}$ .*

By Corollary 16, we observe that NE exists for all  $n$  that admit an existence of a balanced tree. Intuitively, a minor modification of a balanced tree, e.g., removing a subset of leaf nodes, keep the tree stable. Moreover, for  $n \geq 19$  we have found several non-isomorphic equilibrium trees in each case. The number of non-isomorphic equilibria grows with  $n$ , which indicates that for growing  $n$  also the number of possibilities how to combine suitable equilibrium trees into larger equilibrium trees grows. Therefore, we conjecture the existence of stable trees for all values of  $n$  except for  $n = 16$  and  $n = 18$ . We believe that this conjecture can be proven by a dynamic programming approach that exploits the different possibilities of how equilibrium sub-trees can be combined into larger equilibrium trees.

► **Conjecture 1.** *For any  $n \in \mathbb{N}$ , with  $n \neq 16$  and  $n \neq 18$  a pure NE exists in the TCG.*

## 4 Quality of Equilibrium Trees

In this section we provide results on the quality of stable networks. In particular, we prove a constant upper bound on the PoA and give lower bounds on the PoA and PoS. Furthermore, we prove an upper bound on the PoS for certain balanced trees. We first observe that any network in which at least one node has in-degree 2 is not a social optimum. Hence, a Hamilton path is the social optimum.

► **Theorem 17.** *Any Hamiltonian path having the root  $r$  as one endnode is a social optimum.*

### 4.1 Price of Anarchy

In every network  $T$  for all  $v \in V$ ,  $\text{indeg}_T(v) \leq n$ , since there are exactly  $n$  edges. Hence, the cost of an agent is upper bounded by  $n$  and the star graph yields a trivial upper bound of  $n$  for the PoA. However, we prove next a constant upper bound on the PoA.

► **Theorem 18.** *The PoA is at most 8.62.*

**Proof.** Consider a stable network  $T = (V, E)$ . By Theorem 17, the social optimum is a path of cost  $n$ . Hence, it is enough to show that in  $T$  the maximum cost of an agent is upper bounded by a constant. We clearly have that the cost incurred by a non-leaf agent is strictly smaller than the cost incurred by any of its descendants. Therefore, the maximum costs is achieved by a leaf agent.

Consider two leafs  $u$  and  $v$  in  $T$  such that  $u$  pays the maximum cost. Let  $P_v$  be the node-to-root path starting from the parent of  $v$ . Since  $T$  is stable,  $cost_T(u) < 1 + 1/2 + \sum_{(i,j) \in P_v} \frac{indeg(j)}{|T(i)|} = 1/2 + cost_T(v)$ . Therefore, we only have to show that there exists a leaf agent  $v$  with a constant cost value.

We now prove that such a leaf agent always exists. By Corollary 9, each node of in-degree  $d$  has at least  $d - 1$  children of in-degree at least  $\lceil (d - 1)/2 \rceil$ . Consider a root-to-leaf path  $P = (r = v_0, \dots, v_h = v)$  where each next hop goes always towards the smallest appended subtree where the root has an in-degree of at least half of the node's in-degree minus one, i.e., for any  $v_i \in P$ ,  $v_{i+1} = \operatorname{argmin}\{|T(w)| : (w, v_i) \in E \text{ and } indeg(w) \geq (indeg(v_i) - 1)/2\}$ . Then for every  $0 \leq i \leq h - 1$ ,  $|T(v_i)| \geq (indeg(v_i) - 1)|T(v_{i+1})| + 2$ .

Denote by  $|t_k|$  the size of the minimum stable tree with a root of in-degree  $k$ . Then by Corollary 9 and Lemma 6 it holds that

$$|t_0| \geq 1, |t_1| \geq 2, t_k \geq (k - 1) \cdot |t_{\lceil (k-1)/2 \rceil}| + 2. \quad (3)$$

We show via induction that for any  $k \geq 11$ ,  $|t_k| \geq (2k + 1)k^2$ . Indeed, it holds that  $|t_{k+1}| \geq k \cdot |t_{\lceil k/2 \rceil}| + 2 > (k + 1)k^3/2^2 \geq (2(k + 1) + 1)(k + 1)^2$ , where the last inequality holds for all  $k \geq 11$ .

The overall cost incurred by the leaf  $v$  is at most the costs incurred by  $v$  for all edges  $(v_i, v_{i-1}) \in P$  where in-degree of  $v_{i-1}$  is less than the cost incurred by  $v$  for all other edges in  $P$  plus 2.

By Lemma 4, each leaf-to-root path has at most three nodes of in-degree 1, which implies that  $v$  pays at most  $p_1 := \frac{11}{6}$  for all edges ending in a node with in-degree equals 1.

By Lemma 4, the in-degrees of the nodes in the leaf-to-root path  $P$  strictly increase with at least every second hop. This implies that for  $i \leq h - 4 - (11 - 1) \cdot 2 = h - 24$  it is guaranteed that  $indeg(v_i) \geq 11$ . Hence, starting from the first node having in-degree at least 11 in  $P$ , agent  $v$  pays

$$\begin{aligned} p_2 &:= \sum_{i=1}^{h-24} \frac{indeg(v_{i-1})}{|T(v_i)|} \leq \sum_{i=1}^{h-24} \frac{2indeg(v_i) + 1}{|t_{indeg(v_i)}|} \leq \sum_{i=1}^{h-24} \frac{1}{(indeg(v_i))^2} \\ &\leq 2 \cdot \sum_{i=11}^{\infty} \frac{1}{i^2} < 2 \left( \zeta(2) - \sum_{i=1}^{10} \frac{1}{i^2} \right), \end{aligned}$$

where  $\zeta(s)$  is the Riemann zeta function. Hence,  $p_2 < 0.2$ .

Finally, we need to evaluate the cost of the path  $P$  for all nodes  $v_i$  with the in-degree  $2 \leq indeg(v_i) \leq 10$ . Since for every  $0 \leq i \leq h - 1$ ,  $|T(v_i)| \geq (indeg(v_i) - 1)|T(v_{i+1})| + 2$ , each edge  $(v_i, v_{i+1})$  in the path  $P$  costs at most

$$\frac{2indeg(v_i) + 1}{|T(v_i)|} \leq \frac{2indeg(v_i) + 1}{(indeg(v_i) - 1)|T(v_{i+1})|} = \frac{2}{|T(v_{i+1})|} + \frac{3}{(indeg(v_i) - 1)|T(v_{i+1})|}.$$

By applying the inequality (3) and since the in-degrees of the nodes in  $P$  increase at most with every second level, it holds that the total cost of the subpath is at most  $p_3 :=$

## 15:12 Fair Tree Connection Games with Topology-Dependent Edge Cost

$2 \sum_{i=2}^9 \frac{2}{t_i} + \frac{2}{t_1} + \frac{2}{t_{10}} + \sum_{i=2}^{10} \left( \frac{3}{(i-1)t_{i-1}} + \frac{3}{(i-1)t_i} \right) < 3.12 + 2.975 < 6.01$ . Therefore, the total cost of the path  $P$  paid by an agent  $v$  is strictly less than  $p_1 + p_2 + p_3 < 8.12$ . This implies that the PoA is at most 8.62. ◀

We now prove a lower bound to the PoA using the extremal stable balanced trees of Theorem 15. (See Figure 3.) For the rest of this section, let  $\mathcal{T}_h$  denote the extremal balanced tree of height  $h \geq 1$  and degree sequence  $d_h = 0$ ,  $d_{h-1} = 1$ ,  $d_{h-2} = 2$  (if  $h \geq 2$ ),  $d_{h-3} = 4$  (if  $h \geq 3$ ), and  $d_i = 2d_{i+1} + 1$  for every  $i \leq h-4$ . We will denote by  $sc_h$  and  $n_h$  the social cost and the number of nodes (root included) of  $\mathcal{T}_h$ .

► **Theorem 19.** *The PoA is at least 2.4317.*

Next, we prove an upper bound to the average agent's cost in  $\mathcal{T}_h$  and provide an interesting conjecture. We define  $a_h := sc_h / (n_h - 1)$  as the average agent's cost in  $\mathcal{T}_h$ .

► **Lemma 20.** *For every  $h \geq 1$ ,  $a_h \leq 2.4318$ .*

► **Conjecture 2.** *The PoA is equal to  $\lim_{h \rightarrow \infty} a_h$ .*

### 4.2 Price of Stability

We now turn our focus to the PoS and prove a lower bound.

► **Theorem 21.** *The PoS is at least  $\frac{7}{5} - \varepsilon$ , for  $\varepsilon \in \Theta(1/n)$ .*

Next, we investigate the PoS in certain balanced trees and prove an upper bound which is strictly better than the upper bound on the PoA.

► **Theorem 22.** *For all  $n \in \mathbb{N}$  such that there is a balanced tree  $T$  of size  $n$  with the in-degree sequence  $(0, 1, 2, 4, d_{h-4}, \dots, d_0)$ , where  $d_i \leq 2d_{i+1} + 1$  for  $i \leq h-4$ , the PoS is at most 2.83.*

### 4.3 Fairness measure

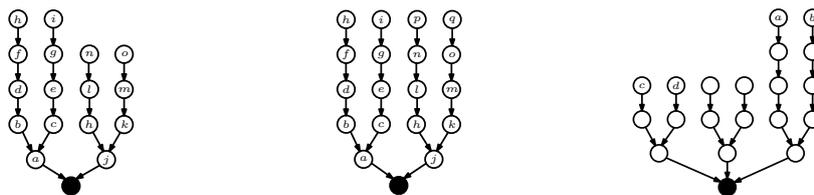
We investigate the Fairness Ratio which considers the cost distribution among the agents. We show that stable trees admit a more fair cost-sharing compared with the social optimum.

► **Theorem 23.** *The Fairness Ratio for  $OPT_n$  is  $nH_n$ , where  $H_n = \sum_{i=1}^n \frac{1}{i}$  is the  $n$ -th harmonic number.*

We now turn our focus to the analysis of the class of all stable trees and prove that the FR is in  $o(n)$ .

► **Theorem 24.** *The Fairness Ratio for any NE is at most  $\frac{8.62(n-2) \cdot \ln \ln(4\sqrt{n/5})}{\ln(4\sqrt{n/5})} - 2^{13}$ .*

$\left(1 - \frac{2 \ln \ln(4\sqrt{n/5})}{\ln(4\sqrt{n/5})}\right) \cdot \left(\frac{\ln(4\sqrt{n/5})}{\ln(4\sqrt{n/5})}\right)^{\log\left(\sqrt{\frac{\ln(4\sqrt{n/5})}{\ln \ln(4\sqrt{n/5})}}\right) - 5.5}$ , which is at most  $8.62 \cdot \frac{(n-2) \cdot \ln \ln(4\sqrt{n/5})}{\ln(4\sqrt{n/5})}$ .



■ **Figure 4** Left: A path-TCG NE that is not a TCG NE for  $n = 16$ . Middle: A path-TCG NE that is not a TCG NE for  $n = 18$ . Right: A TCG NE that is not a strong path-TCG NE.

► **Theorem 25.** *The Fairness Ratio for a stable tree is at least  $n \cdot 2^{-2\sqrt{2\log(n)}}$ .*

Finally, we investigate the class of stable balanced trees and prove a more precise upper bound.

► **Theorem 26.** *The Fairness Ratio for a stable balanced tree with the in-degree sequence  $(0, 1, 2, 4, d_{h-4}, \dots, d_0)$ , where  $d_i \leq 2d_{i+1} + 1$  for  $i \leq h - 4$ , is at most  $\frac{2.4318n \cdot (\ln \ln(4\sqrt{n/5}))^2}{(\ln(4\sqrt{n/5}))^2}$ .*

## 5 Extensions for Future Work: The Path Version and Coalitions

A natural extension of our model is to allow for a richer strategy space. Instead of selecting a single outgoing edge, agents could strategically select a complete path towards the root  $r$ . This version, called the *path-TCG*, is closer to the fair single-source connection game by Anshelevich et al. [5, 4]. See the full version [9] for a formal definition of the path-TCG.

We give some preliminary results relating the equilibria of the TCG to the equilibria of the path-TCG. Our results indicate that studying the path-TCG, in particular its PoA and PoS, is a promising next step. We start with showing that also in the path-TCG all equilibria must be trees.

► **Lemma 27.** *Any equilibrium network in the path-TCG is a tree.*

Now we show that the TCG can be considered as a special case of the path-TCG since all equilibrium trees of the TCG are equilibria in the path-TCG but not vice versa.

► **Theorem 28.** *The set of NE in the path-TCG is a superset of the set of NE in the TCG.*

We showed for the TCG that for  $n = 16$  and  $n = 18$  there exists no stable network. We contrast this negative result with a NE existence proof for the path-TCG for the corresponding values. Figure 4 (left and middle) show equilibrium trees for the path-TCG for  $n = 16$  and  $n = 18$ , respectively.

► **Theorem 29.** *For  $n = 16$  and  $n = 18$  there exists a stable network for the path-TCG.*

Together with Theorem 29 and since any NE in the TCG is a NE in the path-TCG, we go along with Conjecture 1 and believe that for all values of  $n$  stable trees exist for the path-TCG.

► **Conjecture 3.** *For any  $n \in \mathbb{N}$  a pure NE exists in the path-TCG.*

An agent  $a$  in the TCG benefits from the fact that if  $a$  changes her strategy and switches her edge towards another node the costs of the new edge is also shared among all of  $a$ 's ancestors. It seems natural to consider a strategy change in the TCG as a coalitional strategy change in the path-TCG by the coalition consisting of agent  $a$  and all her ancestors. So NE in the TCG could be in strong NE [3] for the pathTCG. However, we show that this is not true, see Figure 4 (right).

► **Theorem 30.** *There is a NE in the TCG which is not in strong NE for the path-TCG.*

## 6 Conclusion

We have studied a tree formation game to investigate how selfish agents self-organize to connect to a common target in the presence of dynamic edge costs that are sensitive to node degrees. This mimics settings in which nodes can charge prices for offering their routing service and where these prices are guided by supply and demand, i.e., more popular nodes with higher in-degree can charge higher prices to make up for their increased internal coordination cost.

Our main findings are that our game admits equilibrium trees with intriguing properties like low height, low maximum degree, almost optimal cost, and a somewhat fair distribution of the total cost among the agents. The set of equilibrium trees seems to be combinatorially rich, and characterizing stable trees that are not balanced seems an exciting and challenging problem for future research. It would also be interesting to study the degree distribution in stable trees and to evaluate possible connections with power-law degree distributions which are ubiquitous in real-world networks.

We note in passing that our model can easily be generalized to settings with more than one target node as long as every possible incident edge may be activated. In this case, several disjoint trees, one for each target node, will be formed. Things change if target nodes and agent nodes may be co-located, and exploring this variant might be interesting.

---

## References

- 1 Susanne Albers and Pascal Lenzner. On Approximate Nash Equilibria in Network Design. *Internet Mathematics*, 9(4):384–405, 2013.
- 2 Carme Àlvarez and Arnau Messegué. On the Price of Anarchy for High-Price Links. In *WINE'19*, pages 316–329. Springer, 2019.
- 3 Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289–317, 2009.
- 4 Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Eva Tardos, Tom Wexler, and Tim Roughgarden. The Price of Stability for Network Design with Fair Cost Allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.
- 5 Elliot Anshelevich, Anirban Dasgupta, Éva Tardos, and Tom Wexler. Near-Optimal Network Design with Selfish Agents. *Theory of Computing*, 4(1):77–109, 2008.
- 6 Venkatesh Bala and Sanjeev Goyal. A Noncooperative Model of Network Formation. *Econometrica*, 68(5):1181–1229, 2000.
- 7 Davide Bilò and Pascal Lenzner. On the Tree Conjecture for the Network Creation Game. *Theory of Computing Systems*, 64(3):422–443, 2020.
- 8 Vittorio Bilò, Michele Flammini, and Luca Moscardelli. The Price of Stability for Undirected Broadcast Network Design with Fair Cost Allocation is Constant. *Games and Economic Behavior*, 2014.
- 9 Davide Bilò, Tobias Friedrich, Pascal Lenzner, Anna Melnichenko, and Louise Molitor. Fair tree connection games with topology-dependent edge cost, 2020. [arXiv:2009.10988](https://arxiv.org/abs/2009.10988).

- 10 Charles G. Bird. On Cost Allocation for a Spanning Tree: A Game Theoretic Approach. *Networks*, 6(4):335–350, 1976.
- 11 Ankit Chauhan, Pascal Lenzner, Anna Melnichenko, and Louise Molitor. Selfish Network Creation with Non-uniform Edge Cost. In *SAGT'17*, pages 160–172. Springer, 2017.
- 12 Chandra Chekuri, Julia Chuzhoy, Liane Lewin-Eytan, Joseph Naor, and Ariel Orda. Non-cooperative Multicast and Facility Location Games. *IEEE Journal on Selected Areas in Communications*, 25(6):1193–1206, 2007.
- 13 Armin Claus and Daniel J. Kleitman. Cost Allocation for a Spanning Tree. *Networks*, 3(4):289–304, 1973.
- 14 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The Price of Anarchy in Network Creation Games. *ACM Transactions on Algorithms*, 8(2):13, 2012.
- 15 Shayan Ehsani, Saber Shokat Fadaee, MohammadAmin Fazli, Abbas Mehrabian, Sina Sadeghian Sadeghabad, Mohammad Ali Safari, and Morteza Saghaian. A Bounded Budget Network Creation Game. *ACM Transactions on Algorithms*, 11(4):1–25, 2015.
- 16 Stephan Eidenbenz, Sritesh Kumar, and Sibylle Zust. Equilibria in Topology Control Games for Ad hoc Networks. *Mobile Networks and Applications*, 11(2):143–159, 2006.
- 17 Bruno Escoffier, Laurent Gourvès, Jérôme Monnot, and Stefano Moretti. Cost Allocation Protocols for Network Formation on Connection Situations. In *ICST'12*, pages 228–234, 2012.
- 18 Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *PODC'03*, pages 347–351. ACM, 2003.
- 19 Michal Feldman, Kevin Lai, and Li Zhang. The Proportional-share Allocation Market for Computational Resources. *IEEE Transactions on Parallel and Distributed Systems*, 20(8):1075–1088, 2008.
- 20 Amos Fiat, Haim Kaplan, Meital Levy, Svetlana Olonetsky, and Ronen Shabo. On the Price of Stability for Designing Undirected Networks with Fair Cost Allocations. In *ICALP'06*, pages 608–618. Springer, 2006.
- 21 Laurent Gourvès and Jérôme Monnot. Three Selfish Spanning Tree Games. In *WINE'08*, pages 465–476. Springer, 2008.
- 22 Daniel Granot and Gur Huberman. Minimum Cost Spanning Tree Games. *Mathematical programming*, 21(1):1–18, 1981.
- 23 Daniel Granot and Gur Huberman. On the Core and Nucleolus of Minimum Cost Spanning Tree Games. *Mathematical programming*, 29(3):323–347, 1984.
- 24 Martin Hoefer. Non-Cooperative Tree Creation. *Algorithmica*, 53(1):104–131, 2009.
- 25 Martin Hoefer and Piotr Krysta. Geometric Network Design with Selfish Agents. In *CO-COON'05*, pages 167–178, 2005.
- 26 Matthew O. Jackson and Asher Wolinsky. A Strategic Model of Social and Economic Networks. *Journal of Economic Theory*, 71(1):44–74, 1996.
- 27 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *STACS'99*, pages 404–413. Springer-Verlag, 1999.
- 28 Thomas L. Magnanti and Richard T. Wong. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*, 18(1):1–55, 1984.
- 29 Akaki Mamageishvili, Matúš Mihalák, and Dominik Müller. Tree Nash Equilibria in the Network Creation Game. In *WAW'13*, pages 118–129. Springer, 2013.
- 30 Matúš Mihalák and Jan Christoph Schlegel. The Price of Anarchy in Network Creation Games is (Mostly) Constant. In *SAGT'10*, pages 276–287. Springer, 2010.
- 31 Kimaya Mittal, Elizabeth M. Belding, and Subhash Suri. A Game-theoretic Analysis of Wireless Access Point Selection by Mobile Users. *Computer Communications*, 31(10):2049–2062, 2008.
- 32 Dov Monderer and Lloyd S. Shapley. Potential Games. *Games and Economic Behavior*, 14(1):124–143, 1996.
- 33 Hervé Moulin and Scott Shenker. Strategyproof Sharing of Submodular Costs: Budget Balance versus Efficiency. *Economic Theory*, 18(3):511–533, 2001.



# Locally Decodable/Correctable Codes for Insertions and Deletions

**Alexander R. Block**

Purdue University, West Lafayette, IN, USA  
block9@purdue.edu

**Jeremiah Blocki**

Purdue University, West Lafayette, IN, USA  
jblocki@purdue.edu

**Elena Grigorescu**

Purdue University, West Lafayette, IN, USA  
elena-g@purdue.edu

**Shubhang Kulkarni<sup>1</sup>**

University of Illinois Urbana-Champaign, IL, USA  
smkulka2@illinois.edu

**Minshen Zhu**

Purdue University, West Lafayette, IN, USA  
zhu628@purdue.edu

---

## Abstract

Recent efforts in coding theory have focused on building codes for insertions and deletions, called insdel codes, with optimal trade-offs between their redundancy and their error-correction capabilities, as well as *efficient* encoding and decoding algorithms.

In many applications, polynomial running time may still be prohibitively expensive, which has motivated the study of codes with *super-efficient* decoding algorithms. These have led to the well-studied notions of Locally Decodable Codes (LDCs) and Locally Correctable Codes (LCCs). Inspired by these notions, Ostrovsky and Paskin-Cherniavsky (Information Theoretic Security, 2015) generalized Hamming LDCs to insertions and deletions. To the best of our knowledge, these are the only known results that study the analogues of Hamming LDCs in channels performing insertions and deletions.

Here we continue the study of insdel codes that admit local algorithms. Specifically, we reprove the results of Ostrovsky and Paskin-Cherniavsky for insdel LDCs using a different set of techniques. We also observe that the techniques extend to constructions of LCCs. Specifically, we obtain insdel LDCs and LCCs from their Hamming LDCs and LCCs analogues, respectively. The rate and error-correction capability blow up only by a constant factor, while the query complexity blows up by a poly log factor in the block length.

Since insdel locally decodable/correctable codes are scarcely studied in the literature, we believe our results and techniques may lead to further research. In particular, we conjecture that constant-query insdel LDCs/LCCs do not exist.

**2012 ACM Subject Classification** Theory of computation → Error-correcting codes

**Keywords and phrases** Locally decodable/correctable codes, insert-delete channel

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.16

**Funding** *Alexander R. Block*: Supported by NSF CCF-1910659.

*Jeremiah Blocki*: Supported by NSF CCF-1910659, CNS-1755708, CNS-1704587 and CNS-1931443.

*Elena Grigorescu*: Supported by NSF CCF-1910659 and NSF CCF-1910411.

*Minshen Zhu*: Supported by NSF CCF-1910659.

---

<sup>1</sup> Work done while at Purdue University, USA.



## 1 Introduction

Building error-correcting codes that can recover from insertions and deletions (a.k.a. “insdel codes”) has been a central theme in recent advances in coding theory [29, 24, 15, 18, 13, 12, 20, 19, 3, 14, 17, 16, 30, 11]. Insdel codes are generalizations of Hamming codes, in which the corruptions may be viewed as deleting symbols and then inserting other symbols at the deleted locations.

An insdel code is described by an encoding function  $E : \Sigma^k \rightarrow \Sigma^n$ , which encodes every message of length  $k$  into a codeword of block length  $n$ . The rate of the code is the ratio  $\frac{k}{n}$ . Classically, a decoding function  $D : \Sigma^* \rightarrow \Sigma^k$  takes as input a string  $w$  obtained from some  $E(m)$  after  $\delta n$  insertions and deletions and satisfies  $D(w) = m$ . A fundamental research direction is building codes with high communication rate  $\frac{k}{n}$ , that are robust against a large  $\delta$  fraction of insertions and deletions, which also admit *efficient* encoding and decoding algorithms. It is only recently that efficient insdel codes with asymptotically good rate and error-correction parameters have been well-understood [17, 19, 16, 30, 11].

In modern applications, polynomial-time decoding may still be prohibitively expensive when working with large data, and instead *super-efficient* codes are even more desirable. Such codes admit very fast decoding algorithms that query only few locations into the received word to recover portions of the data. Ostrovsky and Paskin-Cherniavsky [33] defined the notion of Locally Decodable Insdel Codes,<sup>2</sup> inspired by the notion of Locally Decodable Codes (LDCs) for Hamming errors [22, 36]. A code defined by an encoding  $E : \Sigma^k \rightarrow \Sigma^n$  is a  $q$ -query Locally Decodable Insdel Code (Insdel LDC) if there exists a randomized algorithm  $\mathcal{D}$ , such that: (1) for each  $i \in [k]$  and message  $m \in \Sigma^k$ ,  $\mathcal{D}$  can probabilistically recover  $m_i$ , given query access to a word  $w \in \Sigma^*$ , which was obtained from  $E(m)$  corrupted by  $\delta$  fraction of insertions and deletions; and (2)  $\mathcal{D}$  makes only  $q$  queries into  $w$ . The number of queries  $q$  is called the *locality* of the code.

The rate, error-correcting capability, and locality of the code are opposing design features, and optimizing all of them at the same time is impossible. For example, every 2-query LDCs for Hamming errors must have vanishing rate [23]. While progress in understanding these trade-offs for Hamming errors has spanned several decades [23, 38, 39, 6, 7, 26] (see surveys by Yekhanin [39] and by Kopparty and Saraf [27]), in contrast, the literature on the same trade-offs for the more general insdel codes is scarce. Namely, besides the results of [33], to the best of our knowledge, only Haeupler and Shahrasbi [19] consider the notion of locality in building synchronization strings, which are important components of optimal insdel codes.

The results of [33] provide a direct reduction from classical Hamming error LDCs to insdel LDCs, which preserves the rate of the code and error-correction capabilities up to constant factors, and whose locality grows only by a polylogarithmic factor in the block length.

In this paper we revisit the results of Ostrovsky and Paskin-Cherniavsky [33] and provide an alternate proof, using different combinatorial techniques. We also observe that these results extend to building Locally Correctable Insdel Codes (Insdel LCCs) from Locally Correctable Codes (LCCs) for Hamming errors. LCCs are a variant of LDCs, in which the decoder is tasked to locally correct every entry of the encoded message, namely  $E(m)_i$ , instead of the entries of the message itself. If the message  $m$  is part of the encoding  $E(m)$ , then an LCC is also an LDC. In particular, all linear LCCs (i.e, whose codewords form a vector space) are also LDCs.

---

<sup>2</sup> In [33], they are named *Locally Decodable Codes for Edit Distance*.

► **Theorem 1.** *If there exist  $q$ -query LDCs/LCCs with encoding  $E : \Sigma^k \rightarrow \Sigma^n$ , that can correct from  $\delta$ -fraction of Hamming errors, then there exist binary  $q \cdot \text{polylog}(n)$ -query Insdel LDCs/LCCs with codeword length  $\Theta(n \log |\Sigma|)$ , that can correct from  $\Theta(\delta)$ -fraction of insertions and deletions.*

We emphasize that the resulting LDC/LCC of Theorem 1 is a *binary* code, even if the input LDC/LCC is over some higher alphabet  $\Sigma$ .

Classical constructions of LDCs/LCCs for Hamming errors fall into three query-complexity regimes. In the constant-query regime, the best known results are based on matching-vectors codes, and give encoding that map  $k$  symbols into  $\exp(\exp(\sqrt{\log k \log \log k}))$  symbols [38, 6, 7]. Since the best lower bounds are only quadratic [37], for all we know so far, it is possible that there exist constant-query complexity LDCs with polynomial block length. In the polylog  $k$ -query regime, Reed-Muller codes are examples of  $\log^c k$ -query LDCs/LCCs of block length  $k^{1+\frac{1}{c-1}+o(1)}$  for some  $c > 0$  (e.g., see [39]). Finally, there exist sub-polynomial (but super logarithmic)-query complexity LDCs/LCCs with constant rate [26]. These relatively recent developments improved upon the previous constant rate codes in the  $n^\epsilon$ -query regime achieved by Reed-Muller codes, and later by more efficient constructions (e.g. [28]).

Given that our reduction achieves polylog  $n$ -query complexity blow-up, the results above in conjunction with Theorem 1 give us the following asymptotic results.

► **Corollary 2.** *There exist polylog( $k$ )-query Insdel LDCs/LCCs encoding  $k$  symbols into  $o(k^2)$  symbols, that can correct a constant fraction of insertions and deletions.*

► **Corollary 3.** *There exist  $(\log k)^{O(\log \log k)}$ -query Insdel LDCs/LCCs with constant rate, that can correct from a constant fraction of insertions and deletions.*

Our results, similarly to those in [33], do not have implications in the constant-query regime. We conjecture that there do not exist constant-query LDCs/LCCs, regardless of their rate. Since achieving locality against insertions and deletions appears to be a difficult task, and the area is in its infancy, we believe our results and techniques may motivate further research.

## 1.1 Overview of Techniques

**Searching in a Nearly Sorted Array.** To build intuition for our local decoding algorithm we consider the following simpler problem: We are given a nearly sorted array  $A$  of  $n$  distinct elements. By nearly sorted we mean that there is another sorted array  $A'$  such that  $A'[i] = A[i]$  on all but  $n'$  indices. Given an input  $x$  we would like to quickly find  $x$  in the original array. In the worst case this would require time at least  $\Omega(n')$  so we relax the requirement that we always find  $x$  to say that there are at most  $cn'$  items that we will fail to find for some constant  $c > 0$ .

To design our noisy binary search algorithm that meets the requirement we borrow a notion of local goodness used in the design and analysis of depth-robust graphs a combinatorial object that has found many applications in cryptography [8, 1, 2]. In particular, fixing  $A$  and  $A'$  (sorted) we say that an index  $j$  is corrupted if  $A[j] \neq A'[j]$ . We say that an index  $i$  is  $\theta$ -locally good if for any  $r \geq 0$  at most  $\theta$  fraction of the indices  $j \in [i, \dots, i+r]$  are corrupted and at most  $\theta$  fraction of the indices in  $[i-r, i]$  are corrupted. If at most  $n'$  indices are corrupted then one can prove that at least  $n - 2n'/\theta$  indices are  $\theta$ -locally good [8].

As long as the constant  $\theta$  is suitably small we can design an efficient randomized search procedure which (whp) will correctly locate  $x$  whenever  $x = A[i]$  provided that the unknown index  $i$  is  $\theta$ -locally good. Intuitively, suppose we have already narrowed down our search to the smaller range  $I = [i_0, i_1]$ . The rank of  $x = A[i]$  in  $A'[i_0], \dots, A'[i_1]$  is exactly  $i - i_0 + 1$  since

$A[i]$  is uncorrupted and the rank of  $x$  in  $A[i_0], \dots, A[i_1]$  can change by at most  $\pm\theta(i - i_0 + 1)$  – at most  $\theta(i_1 - i_0 + 1)$  indices  $j' \in [i_0, i_1]$  can be corrupted since  $i \in [i_0, i_1]$  is  $\theta$ -locally good. Now suppose that we sample  $t = \text{polylog}(n)$  indices  $j_1, \dots, j_t \in [i_0, i_1]$  and select the median  $y_{med}$  of  $A[j_1], \dots, A[j_t]$ . With high probability the rank  $r$  of  $y_{med}$  in  $A[j_1], \dots, A[j_t]$  will be close to  $(i_1 - i_0 + 1)/2$  i.e.,  $|r - (i_1 - i_0 + 1)/2| \leq \delta(i_1 - i_0 + 1)$  for some arbitrarily constant  $\delta$  which may depend on the number of samples  $t$ . Thus, for suitable constants  $\theta$  and  $\delta$  whenever  $x > y_{med}$  (resp.  $x < y_{med}$ ) we can safely conclude that  $i > i_0 + (i_1 - i_0 + 1)/8$  (resp.  $i < i_1 - (i_1 - i_0 + 1)/8$ ) and search in the smaller interval  $I' = [i_0 + (i_1 - i_0 + 1)/8, i_1]$  (resp.  $I' = [i_0, i_1 - (i_1 - i_0 + 1)/8]$ ). In both cases the size of the search space is reduced by a constant multiplicative factor so the procedure will terminate after  $O(\log n)$  rounds making  $O(t \log n)$  queries. At its core our local decoding algorithm relies on a very similar idea.

**Encoding.** Our encoder builds off of the known techniques of concatenation codes. First, a message  $x$  is encoded via the outer code to obtain some (intermediate) encoding  $y$ . We then partition  $y$  into some number  $k$  blocks  $y = y_1 \circ \dots \circ y_k$  and append each block  $y_i$  with index  $i$  to obtain  $y_i \circ i$ . Each  $y_i \circ i$  is then encoded with the inner encoder to obtain some  $d_i$ . Then each  $d_i$  is prepended and appended with a run of 0s (i.e., the buffers), to obtain  $c_i$ . The encoder then outputs  $c = c_1 \circ \dots \circ c_k$  as the final codeword. For our inner encoder, we in fact use the Schulman-Zuckerman (SZ) [34] edit distance code.

**Decoding.** Given oracle access to some corrupted codeword  $c'$ , on input index  $i$ , the decoder simulates the outer decoder and must answer the outer decoder oracle queries. The decoder uses the inner decoder to answer these queries. However, there are two major challenges: (1) Unlike the Hamming-type errors, even only a few insertions and deletions makes it hard for the decoder to know where to probe; and (2) The boundaries between blocks can be ambiguous in the presence of insdel errors. We overcome these challenges via a variant of binary search, which we name `NoisyBinarySearch`, together with a buffer detection algorithm, and make use of a block decomposition to facilitate the analysis.

**Analysis.** The analyses of the binary search and the buffer detection algorithms are based on the notion of “good blocks” and “locally good blocks”, which are natural extensions of the notion of  $\theta$ -locally good discussed above. Recall that our encoder outputs a final codeword that is a concatenation of  $k$  smaller codeword “blocks”; namely  $\text{Enc}(x) = c_1 \circ \dots \circ c_k$ . Suppose  $c'$  is the corrupted codeword obtained by corrupting  $c$  with  $\delta$ -fraction of insertion-deletion errors, and suppose we have a method of partitioning  $c'$  into  $k$  blocks  $c'_1 \circ \dots \circ c'_k$ . Then we say that block  $c'_j$  is a  $\gamma$ -good block if it is within  $\gamma$ -fractional edit distance to the *uncorrupted* block  $c_j$ . Moreover,  $c'_j$  is  $(\theta, \gamma)$ -locally good if at least  $(1 - \theta)$  fraction of the blocks in every neighborhood around  $c'_j$  are  $\gamma$ -good and if the total number of corruptions in every neighborhood is bounded. Both notions of good and locally good blocks are necessary to the success of our binary search algorithm `NoisyBinarySearch`.

The goal of `NoisyBinarySearch` is to locate a block with a given index  $j$ , and the idea is to decode the corrupted codeword at random positions to get a list of decoded indices (recall that the index of each block is appended to it). Since a large fraction of blocks are  $\gamma$ -good blocks, the sampled indices induce a new search interval for the next iteration. In order to apply this argument recursively, we need that the error density of the search interval does not increase in each iteration. Locally good blocks provide precisely this property.

**Comparison with the techniques of [33].** The Insdel LDC construction of [33] also uses Schulman-Zuckerman (SZ) [34] codes, except it opens them up and directly uses the inefficient greedy inner codes used for the final efficient SZ codes themselves. In our case, we observe that the efficiently decodable codes of [34] have the additional property described in Lemma 7, which states that small blocks have large weight. This observation implies a running time that is polynomial in the query complexity of the final codes, since it helps make the buffer-finding algorithms local. The analysis of [33] also uses a binary search component, but our analysis and their analysis differ significantly.

## 1.2 Related work

The study of codes for insertions and deletions was initiated by Levenstein [29] in the mid 60's. Since then there has been a large body of works concerning insdel codes, and we refer the reader to the excellent surveys of [35, 31, 32]. In particular, random codes with positive rate correcting from a large fraction of deletions were studied in [24, 15]. Efficiently encodable/decodable codes, with constant rate, and that can withstand a constant fraction of insertion and deletions were extensively studied in [34, 15, 18, 19, 19, 14, 3, 11]. A recent area of interest is building “list-decodable” insdel codes, that can withstand a larger fraction of insertions and deletions, while outputting a small list of potential codewords [20, 11, 30].

In [19], Haeupler and Shahrasbi construct explicit synchronization strings which can be locally decoded, in the sense that each index of the string can be computed using values located at only a small number of other indices. Synchronization strings are powerful combinatorial objects that can be used to index elements in constructions of insdel codes. These explicit and locally decodable synchronizations strings were then used to imply near linear time interactive coding scheme for insdel errors.

There are various other notions of “noisy search” that have been studied in the literature. Dhagat, Gacs, and Winkler [5] consider a noisy version of the game “Twenty Questions”. In this problem, an algorithm searches an array for some element  $x$ , and a bounded number of incorrect answers can be given to the algorithm queries, and the goal is to minimize the number of queries made by an algorithm. Feige et al. [9] study the depth of *noisy decision trees*: decision trees where each node gives the incorrect answer with some constant probability, and moreover each node success or failure is independent. Karp and Kleinberg [21] study noisy binary search where direct comparison between elements is not possible; instead, each element has an associated biased coin. Given  $n$  coins with probabilities  $p_1 \leq \dots \leq p_n$ , target value  $\tau \in [0, 1]$ , and error  $\epsilon$ , the goal is to design an algorithm which, with high probability, finds index  $i$  such that the intervals  $[p_i, p_{i+1}]$  and  $[\tau - \epsilon, \tau + \epsilon]$  intersect. Braverman and Mossel [4], Klein et al. [25] and Geissmann et al. [10] study noisy sorting in the presence of recurrent random errors: when an element is first queried, it has some (independent) probability of returning the incorrect answer, and all subsequent queries to this element are fixed to this answer. We note that each of the above notions of “noisy search” are different from each other and, in particular, different from our noisy search.

## 1.3 Organization

We begin with some general preliminaries in Section 2. In Section 3 we present the formal encoder and decoder. In Section 4 we define block decomposition which play an important role in our analysis. In Section 5, Section 6, and Section 7 we prove correctness of our local decoding algorithm in a top-down fashion. All missing proofs are deferred to the full-version.

## 2 Preliminaries

We now define some notation used throughout this work. We use  $[n]$  to represent the set  $\{1, 2, \dots, n\}$ . More generally, for integers  $a < b$ , we let  $[a, b]$  denote the set  $\{a, a + 1, \dots, b\}$ . All logarithms will be base 2 unless specified otherwise. We denote  $x \circ y$  as the concatenation of string  $x$  with string  $y$ . For any  $x \in \Sigma^n$ ,  $x[i]$  denotes the  $i^{\text{th}}$  coordinate of  $x$ . A function  $\text{negl}(n)$  is said to be *negligible* in  $n$  if  $\text{negl}(n) = o(n^{-d})$  for any  $d \in \mathbb{N}$ . For any  $x, y \in \Sigma^n$ ,  $\text{HAM}(x, y) = |\{i : x[i] \neq y[i]\}|$  denotes the hamming distance between  $x$  and  $y$ . Furthermore,  $\text{ED}(x, y)$  denotes the edit distance between  $x$  and  $y$  i.e. the minimum number of symbol insertions and deletions to transform  $x$  into  $y$ . For any string  $x \in \Sigma^*$  with finite length, we denote  $|x|$  as the length of  $x$ . The fractional Hamming distance (resp., edit distance) is  $\text{HAM}(x, y)/|x|$  (resp.,  $\text{ED}(x, y)/(2|x|)$ ).

► **Definition 4** (Locally Decodable Codes for Hamming and Insdel errors). *A code with encoding function  $E : \Sigma_M^k \rightarrow \Sigma_C^n$  is a  $(q, \delta, \epsilon)$ -Locally Decodable Code (LDC) if there exists a randomized decoder  $\mathcal{D}$ , such that for every message  $m \in \Sigma_M^k$  and index  $i \in [k]$ , and for every  $w \in \Sigma_C^*$  such that  $\text{dist}(w, E(m)) \leq \delta$  the decoder makes at most  $q$  queries to  $w$  and outputs  $m_i$  with probability  $\frac{1}{2} + \epsilon$ ; when  $\text{dist}$  is the fractional Hamming distance then this is a Hamming LDC; when  $\text{dist}$  is the fractional edit distance then this is an Insdel LDC. We also say that the code is binary if  $\Sigma_C = \{0, 1\}$ .*

► **Definition 5** (Locally Correctable Codes for Hamming and Insdel errors). *A code with encoding function  $E : \Sigma_M^k \rightarrow \Sigma_C^n$  is a  $(q, \delta, \epsilon)$ -Locally Correctable Code (LCC) if there exists a randomized decoder  $\mathcal{D}$ , such that for every message  $m \in \Sigma_M^k$  and index  $i \in [n]$ , and for every  $w \in \Sigma_C^*$  such that  $\text{dist}(w, E(m)) \leq \delta$  the decoder makes at most  $q$  queries to  $w$  and outputs  $E(m)_i$  with probability  $\frac{1}{2} + \epsilon$ ; when  $\text{dist}$  is the fractional Hamming distance then this is a Hamming error LCC; when  $\text{dist}$  is the fractional edit distance then this is an Insdel LCC. We also say that the code is binary if  $\Sigma_C = \{0, 1\}$ .*

Our construction, like most insdel codes in the literature, is obtained via adaptations of the simple but powerful operation of code concatenation. If  $C_{out}$  is an “outer” code over alphabet  $\Sigma_{out}$  with encoding function  $E_{out} : \Sigma_{out}^k \rightarrow \Sigma_{out}^n$ , and  $C_{in}$  is an “inner” code over alphabet  $\Sigma_{in}$  with encoding function  $E_{in} : \Sigma_{in}^p \rightarrow \Sigma_{in}^p$ , then the concatenated code  $C_{out} \bullet C_{in}$  is the code whose codewords lie in  $\Sigma_{in}^{np}$ , obtained by first applying  $E_{out}$  to the message, and then applying  $E_{in}$  to each symbol of the resulting outer codeword.

## 3 Insdel LDCs/LCCs from Hamming LDCs/LCCs

We give our main construction of Insdel LDCs/LCCs from Hamming LDCs/LCCs. Our construction can be viewed as a procedure which, given outer codes  $C_{out}$  and binary inner codes  $C_{in}$  satisfying certain properties, produces binary codes  $C(C_{out}, C_{in})$ . This is formulated in the following theorem, which implies Theorem 1.

- **Theorem 6.** *Let  $C_{out}$  and  $C_{in}$  be codes such that*
- *$C_{out}$  defined by  $\text{Enc}_{out} : \Sigma^k \rightarrow \Sigma^m$  is an a  $(\ell_{out}, \delta_{out}, \epsilon_{out})$ -LDC/LCC (for Hamming errors).*
  - *$C_{in}$  is family of binary polynomial-time encodable/decodable codes with rate  $1/\beta_{in}$  capable of correcting  $\delta_{in}$  fraction of insdel errors. In addition, there are constants  $\alpha_1, \alpha_2 \in (0, 1)$  such that for any codeword  $c$  of  $C_{in}$ , any substring of  $c$  with length  $\geq \alpha_1|c|$  has fractional Hamming weight at least  $\alpha_2$ .*

Then  $C(C_{out}, C_{in})$  is a binary  $\left(\ell_{out} \cdot O\left(\log^4 n'\right), \Omega(\delta_{out}\delta_{in}), \epsilon - \text{negl}(n')\right)$ -Insdel LDC, or a binary  $\left(\ell_{out} \cdot O\left(\log^5 n'\right), \Omega(\delta_{out}\delta_{in}), \epsilon - \text{negl}(n')\right)$ -Insdel LCC, respectively. Here the codewords of  $C$  have length  $n = \beta m$  where  $\beta = O\left(\beta_{in} \log |\Sigma|\right)$ , and  $n'$  denotes the length of received word.

For the inner code, we make use of the following efficient code constructed by Schulman-Zuckerman [34].

► **Lemma 7** (SZ-code [34]). *There exist constants  $\beta_{in} \geq 1$ ,  $\delta_{in} > 0$ , such that for large enough values of  $t > 0$ , there exists a code  $SZ(t) = (\text{Enc}, \text{Dec})$  where  $\text{Enc} : \{0, 1\}^t \rightarrow \{0, 1\}^{\beta_{in} t}$  and  $\text{Dec} : \{0, 1\}^{\beta_{in} t} \rightarrow \{0, 1\}^t \cup \{\perp\}$  capable of correcting  $\delta_{in}$  fraction of insdel errors, having the following properties:*

1.  $\text{Enc}$  and  $\text{Dec}$  run in time  $\text{poly}(t)$ ;
2. For all  $x \in \{0, 1\}^t$ , every interval of length  $2 \log t$  of  $\text{Enc}(x)$  has fractional Hamming weight  $\geq 2/5$ .

We formally complete the proof of correctness of Theorem 6 in Section 5. We only prove the correctness of the LDC decoder since it is cleaner and captures the general strategy of the LCC decoder as well. We dedicate the remainder of this section to outlining the construction of the encoding and decoding algorithms.

### 3.1 Encoding and Decoding Algorithms

In our construction of  $C(C_{out}, C_{in})$ , we denote the specific code of Lemma 7 as our inner code  $C_{in} = (\text{Enc}_{in}, \text{Dec}_{in})$ . For our purpose, it is convenient to view the message as a pair in  $[m] \times \Sigma^{\log m}$ . The encoding function  $\text{Enc}_{in} : [m] \times \Sigma^{\log m} \rightarrow \{0, 1\}^{\beta_{in}(1+\log|\Sigma|)\log m}$  maps a  $\log m$ -long string over alphabet  $\Sigma$  appended with an index from set  $[m]$  – i.e. a (padded) message of bit-length  $(1 + \log |\Sigma|) \log m$  – to a binary string of length  $\beta_{in}(1 + \log |\Sigma|) \log m$ . The inner decoder  $\text{Dec}_{in}$  on input  $y'$  returns  $x$  if  $\text{ED}(y', y) \leq \delta_{in} \cdot 2|y|$  where  $y = \text{Enc}_{in}(x)$ . The information rate of this code is  $R_{in} = 1/\beta_{in}$ .

We describe our final encoding and decoding algorithms next.

**The Encoder (Enc).** Given an input string  $x \in \Sigma^k$  and outer code  $C_{out} = (\text{Enc}_{out}, \text{Dec}_{out})$ , our final encoder  $\text{Enc}$  does the following: (1) Computes the outer encoding of  $x$  as  $s = \text{Enc}_{out}(x)$ ; (2) For each  $i \in [m/\log m]$ , groups  $\log m$  symbols  $s[(i-1)\log m] \cdots s[i\log m - 1]$  into a single block  $b_i \in \Sigma^{\log m}$ ; (3) For each  $i \in [m/\log m]$ , computes the  $i^{\text{th}}$  block of the inner encoding as  $Y^{(i)} = \text{Enc}_{in}(i \circ b_i)$  – i.e. computes inner encoding of the  $i^{\text{th}}$  block concatenated with the (padded) index  $i$ ; (4) For some constant  $\alpha \in (0, 1)$  (to be decided), appends a  $\alpha \log m$ -long buffer of zeros before and after each block; and (5) Outputs the concatenation of the buffered blocks (in indexed order) as the final codeword  $c = \text{Enc}(x) \in \{0, 1\}^n$ , where

$$c = \left(0^{\alpha \log m} \circ Y^{(1)} \circ 0^{\alpha \log m}\right) \circ \dots \circ \left(0^{\alpha \log m} \circ Y^{(m/\log m)} \circ 0^{\alpha \log m}\right). \quad (1)$$

Denoting  $\beta = 2\alpha + \beta_{in}(1 + \log |\Sigma|)$ , the length of  $c$  is

$$n = \left(2\alpha \log m + \beta_{in}(1 + \log |\Sigma|) \log m\right) \cdot \frac{m}{\log m} = \beta m.$$

**The LDC Decoder (Dec).** We start off by describing the high-level overview of our decoder Dec and discuss the challenges and solutions behind its design. As defined in Equation (1), our encoder Enc, on input  $x \in \Sigma^k$ , outputs a codeword  $c = c_1 \circ \dots \circ c_d \in \{0, 1\}^n$  where  $d = m/\log m$ . The decoder setting is as follows: on input  $i \in [k]$  and query access to the corrupted codeword  $c' \in \{0, 1\}^{n'}$  such that  $\text{ED}(c, c') \leq 2n\delta$ , our final decoder Dec needs to output the message symbol  $x[i]$  with high probability. Notice that if Dec had access to the original codeword  $s = \text{Enc}_{out}(x)$ , then Dec could simply run  $\text{Dec}_{out}(i)$  while supplying it with oracle access to this codeword  $s$ . This naturally motivates the following decoding strategy – simulate  $\text{Dec}_{out}$ 's oracle access to the codeword  $s$  via answering  $\text{Dec}_{out}$ 's queries by decoding the appropriate bits using  $\text{Dec}_{in}$ . We give a detailed description of this strategy next.

Let  $Q_i = \{q_1, \dots, q_{\ell_{out}}\} \subset [m]$  be a set of indices which  $\text{Dec}_{out}(i)$  makes queries to.<sup>3</sup> We observe that if our decoder had oracle access to the uncorrupted codeword  $c$ , then answering these queries would be simple:

1. For each  $q \in Q_i$ , let  $b_j = s[(j-1)\log m] \dots s[j\log m - 1]$  be the block which contains  $s[q]$ . In particular,  $q = (j-1)\log m + r_j$  for some  $r_j \in [0, \log m - 1]$ ,
2. Obtain block  $c_j$  by querying oracle  $c$  and obtain  $Y^{(j)}$  by removing the buffers from  $c_j$ ,
3. Obtain  $j \circ b_j$  by running  $\text{Dec}_{in}(Y^{(j)})$ , then return  $s[q] = b_j[r_j]$  to  $\text{Dec}_{out}$ .

In fact, it suffices for  $\text{Dec}_{out}$ 's queries to be answered with symbols consistent with any string  $s'$  such that  $\text{HAM}(s, s') \leq \delta_{out}m$  – for then, the correctness of the output would follow from the correctness of  $\text{Dec}_{out}$ . We are still going to carry out the strategy mentioned above, except that now we are given a corrupted codeword  $c'$ .

For the purposes of analysis, we first define the notion of a *block decomposition* of the corrupted codeword  $c'$ . Informally, a block decomposition is simply a partitioning of  $c'$  into contiguous blocks. Our first requirement for successful decomposition is that there must exist a block decomposition  $c' = c'_1 \circ \dots \circ c'_d$  that is “not too different” from the original decomposition  $c = c_1 \circ \dots \circ c_d$ .<sup>4</sup> In particular, we require that  $\sum_j \text{ED}(c'_j, c_j) \leq 2n\delta$ . Proposition 9 guarantees this property. Next, we define the notion of  $\gamma$ -good (see Definition 10). The idea here is that if a block  $c'_j$  is  $\gamma$ -good (for appropriate  $\gamma$ ), then we can run  $\text{Dec}_{in}$  on  $c'_j$  and obtain  $j \circ b_j$ . As the total number of errors is bounded, it is easy to see that all but a small fraction of blocks are  $\gamma$ -good (Lemma 14). At this point, we are essentially done if we can decode  $c'_j$  for any given  $\gamma$ -good block  $j$ .

An immediate challenge we are facing is that of *locating* a specific  $\gamma$ -good block  $c'_j$ , while maintaining overall locality. The presence of insertions and deletions may result in uneven block lengths, making the task of locating a specific block non-trivial. However, since the  $\gamma$ -good blocks, which make up majority of the blocks, must be in the correct relative order, it is conceivable to perform a *binary search* type algorithm over the blocks of  $c'$  to find block  $c'_j$ . The idea is to maintain a search interval and iteratively reduce its size. In each iteration, the algorithm samples a small number of blocks and obtains their (appended) indices. As the vast majority of blocks are  $\gamma$ -good, these indices will guide the binary search algorithm in narrowing down the search interval. There is one problem with this argument. The density of  $\gamma$ -good blocks may go down as the search interval gets smaller. In fact, it is impossible to *locally* locate a block  $c'_j$  surrounded by many bad blocks even if it is  $\gamma$ -good. This is where the notion of  $(\theta, \gamma)$ -locally good (see Definition 12) helps us: if a block  $c'_j$  is  $(\theta, \gamma)$ -locally

<sup>3</sup> This is for ease of presentation. Our construction also supports adaptive queries.

<sup>4</sup> We note that we do not need to know this decomposition explicitly, and that its existence is sufficient for our analysis.

good, then  $(1 - \theta)$ -fraction of blocks in every neighborhood around  $c'_j$  are  $\gamma$ -good, and every neighborhood around  $c'_j$  has a bounded number of errors. Therefore, as long as the search interval contains a locally good block, we can lower bound the density of  $\gamma$ -good blocks.

Our *noisy binary search algorithm* essentially implements this idea. The algorithm on input block index  $j$ , attempts to find block  $j$ . If block  $j$  is  $(\theta, \gamma)$ -locally good, then we can guarantee that our noisy binary search algorithm will find  $j$  except with negligible probability (see Theorem 18). Thus it is desirable that the number of  $(\theta, \gamma)$ -locally good blocks is large; if this is so, the noisy binary search is effectively providing oracle access to a string  $s'$  which is close to  $s$  in Hamming distance, and thus the outer decoder is able to decode  $x[i]$  with high probability. Lemma 15 exactly guarantees this property.

The discussion above, however, requires knowing the boundaries of each block  $c'_j$ . As the decoder is oblivious to the block decomposition, it is going to work with *approximate boundaries* which can be found locally by a *buffer search algorithm* described as follows. Recall that by construction  $c_j$  consists of  $Y^{(j)}$  surrounded by buffers of  $(\alpha \log m)$ -length 0-runs. So to find  $Y^{(j)}$ , it suffices to find the buffers surrounding  $Y^{(j)}$ . Our buffer search algorithm can be viewed as a “local variant” of the buffer search algorithm of Schulman and Zuckerman [34]. It is designed to find approximate buffers surrounding a block  $c'_j$  if it is  $\gamma$ -good. Then the string in between two buffers is identified as a corrupted codeword and is decoded to  $j \circ b_j$ . The success of the algorithm depends on  $\gamma$ -goodness of the block being searched and requires that any substring of a codeword from  $C_{in}$  has “large enough” Hamming weight. In fact, our inner code given by Lemma 7 gives us this exact guarantee. All together, this enables the noisy binary search algorithm to use the buffer finding algorithm to search for a block  $c'_j$ .

We formalize the decoder outlined above. On input  $i \in [k]$ , Dec simulates  $\text{Dec}_{out}(i)$  and answers its queries. Whenever  $\text{Dec}_{out}(i)$  queries an index  $j \in [m]$ , Dec expresses  $j = (p - 1) \log m + r_j$  for  $p \in [m / \log m]$  and  $r_j \in [0, \log m - 1]$ , and runs  $\text{NoisyBinarySearch}(c', p)$  (which calls the  $\text{BUFF-FIND}$  algorithm) to obtain a string  $b' \in \Sigma^{\log m}$  (or  $\perp$ ). Then it feeds the  $r_j$ -th symbol of  $b'$  (or  $\perp$ ) to  $\text{Dec}_{out}(i)$ . Finally, Dec returns the output of  $\text{Dec}_{out}(i)$ .

**The LCC Decoder (Dec).** Similar to the LDC decoder, our LCC decoder Dec does the following: We denote  $B = 2\alpha \log m + \beta_{in} (1 + \log |\Sigma|) \log m$ . On input  $j \in [n]$ , Dec first expresses  $j = (p - 1)B + r_j$  for some  $p \in [m / \log m]$  and  $0 \leq r_j < B$ , and checks whether  $j$  is inside a buffer. Specifically, if  $r_j \in [0, \log m) \cup [B - \log m, B)$ , it outputs 0. Otherwise, it simulates  $\text{Dec}_{out}((p-1) \log m + r)$  for each  $0 \leq r < \log m$ , and answers their queries. Whenever  $\text{Dec}_{out}$  queries  $i \in [m]$ , Dec expresses  $i = (b - 1) \log m + r_i$  for some  $b \in [m / \log m]$  and  $0 \leq r_i < \log m$ , and runs  $\text{NoisyBinarySearch}(c', b)$  to obtain a string  $S \in \Sigma^{\log m}$  (or  $\perp$ ), and answers the query with  $S_{r_i}$  (or  $\perp$ ). Finally, denoting by  $s_r$  the output of  $\text{Dec}_{out}((p - 1) \log m + r)$ , Dec returns the  $(r_j - \log m + 1)$ -th bit of  $\text{Enc}_{in}(p \circ s_0 s_1 \dots s_{\log m - 1})$ .

**Efficiency.** We note that the efficiency of our compiler depends on the constituent inner and outer codes. Let  $T(\text{Enc}_{in}, l)$ ,  $T(\text{Enc}_{out}, l)$ ,  $T(\text{Enc}, l)$  denote the run-times of the inner, outer and final encoders respectively on inputs of length  $l$ . Similarly, let  $T(\text{Dec}_{in}, l)$ ,  $T(\text{Dec}_{out}, l)$ ,  $T(\text{Dec}, l)$  denote the run-times of the inner, outer and final decoders respectively having query access to corrupted codewords of length  $l$ . Then we have following run-time relations

$$\begin{aligned} T(\text{Enc}, k) &= T(\text{Enc}_{out}, k) + O(m / \log m) \cdot T(\text{Enc}_{in}, \log |\Sigma| \cdot \log m + \log m), \\ T(\text{Dec}, n') &= T(\text{Dec}_{out}, m) + \ell_{out} \cdot O(\log^3 n') \cdot T(\text{Dec}_{in}, \beta \log m). \end{aligned}$$

Here,  $n'$  is the length of the corrupted codeword.

#### 4 Block Decomposition of Corrupted Codewords

The analysis of our decoding procedure relies on a so-called buffer finding algorithm and a noisy binary search algorithm. To analyze these algorithms, we introduce the notion of a *block decomposition* for (corrupted) codewords, as well as what it means for a block to be (*locally*) *good*.

For convenience, we now fix some notations for the rest of the paper. We fix an arbitrary message  $x \in \Sigma^k$ . We use  $s = \text{Enc}_{out}(x) \in \Sigma^m$  for the encoding of  $x$  by the outer encoder. Let  $\tau = \log m$  be the length of each block and  $d = m/\log m$  be the number of blocks. For  $i \in [d]$ , we let  $b_i \in \Sigma^\tau$  denote the  $i$ -th block  $s[(i-1)\tau]s[(i-1)\tau+1] \dots s[i\tau-1]$ , and let  $Y^{(i)}$  denote the encoding  $\text{Enc}_{in}(i \circ b_i)$ . Recall that  $\alpha\tau$  is the length of the appended buffers for some  $\alpha \in (0, 1)$ , and the parameter  $\beta = 2\alpha + \beta_{in}(1 + \log |\Sigma|)$ . Thus  $|Y^{(i)}| = (\beta - 2\alpha)\tau$ . The final encoding is given by

$$c = \tilde{Y}^{(1)} \circ \tilde{Y}^{(2)} \circ \dots \circ \tilde{Y}^{(d)},$$

where  $\tilde{Y}^{(j)} = 0^{\alpha\tau} \circ Y^{(j)} \circ 0^{\alpha\tau}$  and  $|\tilde{Y}^{(j)}| = \beta\tau$ . The length of  $c$  is  $n = d\beta\tau = \beta m$ . We let  $c' \in \{0, 1\}^{n'}$  denote a corrupted codeword satisfying  $\text{ED}(c, c') \leq \delta \cdot 2n$ .

A *block decomposition* of a (corrupted) codeword  $c'$  is a non-decreasing mapping  $\phi: [n'] \rightarrow [d]$  for  $n', d \in \mathbb{Z}^+$ . We say a set  $I \subseteq [n']$  is an interval if  $I = \emptyset$  (i.e., an empty interval) or  $I = \{l, l+1, \dots, r-1\}$  for some  $1 \leq l < r \leq n'$ , in which case we write  $I = [l, r)$ . For an interval  $I = [l, r)$ , we write  $c'[I]$  for the substring  $c'[l]c'[l+1] \dots c'[r-1]$ . Finally,  $c[\emptyset]$  stands for the empty string.

We remark that since  $\phi$  is non-decreasing, for every  $j \in [d]$  the pre-image  $\phi^{-1}(j)$  is an interval. Since  $\phi$  is a total function, it induces a partition of  $[n']$  into  $d$  intervals  $\{\phi^{-1}(j) : j \in [d]\}$ . The following definition plays an important role in the analysis.

► **Definition 8** (closure intervals). *The closure of an interval  $I = [l, r) \subseteq [n']$  is defined as  $\bigcup_{i=l}^{r-1} \phi^{-1}(\phi(i))$ . An interval  $I$  is a closure interval if the closure of  $I$  is itself. Equivalently, every closure interval has the form  $\mathcal{I}[a, b] := \bigcup_{j=a}^b \phi^{-1}(j)$  for some  $a, b \in [d]$ .*

► **Proposition 9**. *There exists a block decomposition  $\phi: [n'] \rightarrow [d]$  such that*

$$\sum_{j \in [d]} \text{ED}(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}) \leq \delta \cdot 2n.$$

We now introduce the notion of *good blocks*. In the following definitions, we also fix an arbitrary block decomposition  $\phi$  of  $c'$  enjoying the property guaranteed by Proposition 9.

► **Definition 10** ( $\gamma$ -good block). *For  $\gamma \in (0, 1)$  and  $j \in [d]$  we say that block  $j$  is  $\gamma$ -good if  $\text{ED}(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}) \leq \gamma\alpha\tau$ . Otherwise we say that block  $j$  is  $\gamma$ -bad.*

► **Definition 11** ( $(\theta, \gamma)$ -good interval). *We say a closure interval  $\mathcal{I}[a, b]$  is  $(\theta, \gamma)$ -good if the following hold:*

1.  $\sum_{j=a}^b \text{ED}(c'[\phi^{-1}(j)], \tilde{Y}^{(j)}) \leq \gamma \cdot (b - a + 1)\alpha\tau$ .
2. *There are at least  $(1-\theta)$ -fraction of  $\gamma$ -good blocks among those indexed by  $\{a, a+1, \dots, b\}$ .*

► **Definition 12** ( $(\theta, \gamma)$ -local good block). *For  $\theta, \gamma \in (0, 1)$  we say that block  $j$  is  $(\theta, \gamma)$ -local good if for every  $a, b \in [d]$  such that  $a \leq j \leq b$  the interval  $\mathcal{I}[a, b]$  is  $(\theta, \gamma)$ -good. Otherwise, block  $j$  is  $(\theta, \gamma)$ -locally bad.*

Note that in Definition 12, if  $j$  is  $(\theta, \gamma)$ -locally good, then  $j$  is also  $\gamma$ -good by taking  $a = b = j$ .

► **Proposition 13.** *The following bounds hold:*

1. For any  $\gamma$ -good block  $j$ ,  $(\beta - \alpha\gamma)\tau \leq |\phi^{-1}(j)| \leq (\beta + \alpha\gamma)\tau$ .
2. For any  $(\theta, \gamma)$ -good interval  $\mathcal{I}[a, b]$ ,  $(b - a + 1)(\beta - \alpha\gamma)\tau \leq |\mathcal{I}[a, b]| \leq (b - a + 1)(\beta + \alpha\gamma)\tau$ .

The following lemmas give upper bounds on the number of  $\gamma$ -bad and  $(\theta, \gamma)$ -locally bad blocks.

► **Lemma 14.** *The total fraction of  $\gamma$ -bad blocks is at most  $2\beta\delta/(\gamma\alpha)$ .*

► **Lemma 15.** *The total fraction of  $(\theta, \gamma)$ -local bad blocks is at most  $(4/\gamma\alpha)(1 + 1/\theta)\delta\beta$ .*

## 5 Outer Decoder

At a high level, the our decoding algorithm Dec the outer decoder  $\text{Dec}_{out}$  and must answer all oracle queries of  $\text{Dec}_{out}$  by simulating oracle access to some corrupted string  $s'$ . Recall that  $C_{out}$  with encoding function  $\text{Enc}_{out}: \Sigma^k \rightarrow \Sigma^m$  is a  $(\ell_{out}, \delta_{out}, \epsilon_{out})$ -LDC (for Hamming errors). There is a probabilistic decoder  $\text{Dec}_{out}$  such that for any  $i \in [k]$  and string  $s' \in (\Sigma \cup \{\perp\})^m$  such that  $\text{HAM}(s', s) \leq \delta_{out} \cdot m$  for some codeword  $s = \text{Enc}_{out}(x)$ , we have

$$\Pr \left[ \text{Dec}_{out}^{s'}(i) = x[i] \right] \geq \frac{1}{2} + \epsilon_{out}.$$

Additionally,  $\text{Dec}_{out}$  makes at most  $\ell_{out}$  queries to  $s'$ .

In order to run  $\text{Dec}_{out}$ , we need to simulate oracle access to such a string  $s'$ . To do so, we present our noisy binary search algorithm Algorithm 1 in Section 6. For now, we assume Algorithm 1 has the property stated in the following theorem.

► **Theorem 16.** *For  $j \in [d]$ , let  $\mathbf{b}_j \in \Sigma^\tau \cup \{\perp\}$  be the random variable denoting the output of Algorithm 1 on input  $(c', 1, n' + 1, j)$ . We have*

$$\Pr \left[ \Pr_{j \in [d]} [\mathbf{b}_j \neq b_j] \geq \delta_{out} \right] \leq \text{negl}(n'),$$

where the probability is taken over the joint distribution of  $\{\mathbf{b}_j: j \in [d]\}$ .

We note that  $\mathbf{b}_j$ 's do not need to be independent, i.e. two runs of Algorithm 1 can be correlated. For example, we can fix the random coin tosses of Algorithm 1 before the first run and reuse them in each call.

## 6 Noisy Binary Search

We present Algorithm 1 below. As mentioned in Section 5, the binary search algorithm discussed in this section can be viewed as providing the outer decoder with oracle access to some string  $s' \in (\Sigma \cup \{\perp\})^m$ . Namely whenever the outer decoder queries an index  $j \in [m]$  which lies in block  $p$ , we run NOISY-BINARY-SEARCH on  $(c', 1, n' + 1, p)$  and obtain a string  $b'_p \in \Sigma^{\log m}$  which contains the desired symbol  $s'[j]$ . For now, we analyze the query complexity of Algorithm 1.

► **Proposition 17.** *Algorithm 1 has query complexity  $O(\log^4 n')$ .*

The following theorem shows that the set of indices which can be correctly returned by Algorithm 1 is captured by the locally good property.

---

**Algorithm 1** NOISY BINARY SEARCH.
 

---

**Input:** An index  $j \in [d]$ , and oracle access to a codeword  $c' \in \{0, 1\}^{n'}$ .

**Output:** A string  $b \in \Sigma^\tau$  or  $\perp$ .

```

1:  $N \leftarrow \Theta(\log^2 n')$ 
2:  $\rho \leftarrow \min \left\{ \frac{1}{4} \cdot \frac{\beta - \gamma}{\beta + \gamma}, 1 - \frac{3}{4} \cdot \frac{\beta + \gamma}{\beta - \gamma} \right\}$ 
3:  $C \leftarrow 36(\beta + \gamma)\tau$ 
4: function NOISY-BINARY-SEARCH( $c', l, r, j$ )
5:   if  $r - l \leq C$  then
6:      $s \leftarrow$  INTERVAL-DECODE( $l, r, j$ )
7:     return  $s$ 
8:   end if
9:    $m_1 \leftarrow (1 - \rho)l + \rho r, m_2 \leftarrow \rho l + (1 - \rho)r$ 
10:  for  $t \leftarrow 1$  to  $N$  do
11:    Randomly sample  $i$  from  $\{m_1, m_1 + 1, \dots, m_2 - 1\}$ 
12:     $j_t \leftarrow$  BLOCK-DECODE( $i$ )
13:  end for
14:   $\tilde{j} \leftarrow$  median of  $j_1, \dots, j_N$  (ignore  $j_t$  if  $j_t = \perp$ )
15:  if  $j \leq \tilde{j}$  then
16:    return NOISY-BINARY-SEARCH( $c', l, m_2, j$ )
17:  else
18:    return NOISY-BINARY-SEARCH( $c', m_1, r, j$ )
19:  end if
20: end function

```

---

► **Theorem 18.** *If  $j \in [d]$  is a  $(\theta, \gamma)$ -locally-good block, running Algorithm 1 on input  $(c', 1, n' + 1, j)$  outputs  $b_j$  with probability at least  $1 - \text{negl}(n')$ .*

As the only time Algorithm 1 interacts with  $c'$  is when it queries BLOCK-DECODE and INTERVAL-DECODE, the properties of these two algorithms are going to be essential to our proof. We briefly describe these two subroutines now.

- **BLOCK-DECODE** On input index  $i \in [n']$ , BLOCK-DECODE tries to find the block  $j$  that contains  $i$ , and attempts to decode the block to  $j \circ b_j$ . It returns the index  $j$  if the decoding was successful, and  $\perp$  otherwise.
- **INTERVAL-DECODE** On input  $l, r \in [n']$  and  $j \in [d]$ , INTERVAL-DECODE (roughly) runs the buffer search algorithm of Schulman and Zuckerman [34] over the substring  $c'[l, r]$  to obtain a set of approximate buffers, and attempts to decode all strings separated by the approximate buffers. It returns  $b$  if any string is decoded to  $j \circ b$ , and  $\perp$  otherwise.

For convenience, we will model BLOCK-DECODE as a function  $\varphi: [n'] \rightarrow [d] \cup \{\perp\}$ , and model INTERVAL-DECODE as a function  $\psi: [n'] \rightarrow \Sigma^\tau \cup \{\perp\}$ . The following properties of  $\varphi$  and  $\psi$  are what allow the proof to go through.

► **Theorem 19.** *The functions  $\varphi$  and  $\psi$  satisfy the following properties:*

1. *For any  $\gamma$ -good block  $j$  we have*

$$\Pr_{i \in \phi^{-1}(j)} [\varphi(i) \neq j] \leq \gamma.$$

2. *Let  $[l, r)$  be an interval with closure  $\mathcal{I}[L, R - 1]$  such that every block  $j \in \{L, \dots, R - 1\}$  is  $\gamma$ -good. Then for every block  $j$  such that  $\phi^{-1}(j) \subseteq [l, r)$ , we have  $\psi(j, l, r) = b_j$ .*

## 7 Block Decode Algorithm

A key component of the Noisy Binary Search algorithm is the ability to decode  $\gamma$ -good blocks in the corrupted codeword  $c'$ . In order to do so, our algorithm will take explicit advantage of the  $\gamma$ -good properties of a block. We present our block decoding algorithm, named BLOCK-DECODE, in Algorithm 2.

### Algorithm 2 BLOCK-DECODE.

**Input:** An index  $i \in [n']$  and oracle access to (corrupted) codeword  $c' \in \{0, 1\}^{n'}$ .

**Output:** Some string  $\text{Dec}(s)$  for a substring  $s$  of  $c'$ , or  $\perp$ .

```

1: function BLOCK-DECODE $^{c'}$ ( $i$ )
2:   buff  $\leftarrow$  BUFF-FIND $_{\eta}^{c'}$ ( $i$ )
3:   if buff ==  $\perp$  then
4:     return  $\perp$ 
5:   else Parse buff as  $(a, b), (a', b')$ 
6:     if  $b < i < a'$  then
7:       return  $\text{Dec}_{in}(c'[b+1, a'-1])$ 
8:     end if
9:   end if
10:  return  $\perp$ 
11: end function

```

### Algorithm 3 BUFF-FIND $_{\eta}$ .

**Input:** An index  $i \in [n']$  and oracle access to (corrupted) codeword  $c' \in \{0, 1\}^{n'}$ .

**Output:** Two consecutive  $\delta_b$ -approximate buffers  $(a, b), (a', b')$ , or  $\perp$ .

```

1: function BUFF-FIND $^{c'}$ ( $i$ )
2:    $j_s \leftarrow \max\{1, i - \eta\tau\}, j_e \leftarrow \min\{n' - \tau + 1, i + \eta\tau\}$ 
3:   buffs  $\leftarrow []$ 
4:   while  $j_s \leq j_e$  do
5:     if  $\text{ED}(0^{\tau}, c'[j_s, j_s + \tau - 1]) \leq \delta_b \alpha \tau$  then
6:       buffs.append( $(j_s, j_s + \tau - 1)$ )
7:     end if
8:      $j_s \leftarrow j_s + 1$ 
9:   end while
10:  for all  $k \in \{0, 1, \dots, |\text{buffs}| - 2\}$  do
11:     $(a, b) \leftarrow \text{buffs}[k], (a', b') \leftarrow \text{buffs}[k + 1]$ 
12:    if  $b < i < a'$  then
13:      return  $(a, b), (a', b')$ 
14:    end if
15:  end for
16:  return  $\perp$ 
17: end function

```

### 7.1 Buff-Find

The algorithm BLOCK-DECODE makes use of the sub-routine BUFF-FIND, presented in Algorithm 3. At a high-level, the algorithm BUFF-FIND on input  $i$  and given oracle access

to (corrupted) codeword  $c'$  searches the ball  $c'[i - \eta\tau, i + \eta\tau]$  for all  $\delta_b$ -approximate buffers in the interval, where  $\eta \geq 1$  is a constant such that if  $i \in \phi^{-1}(j)$  for any good block  $j$  then  $c'[\phi^{-1}(j)] \subseteq c'[i - \eta\tau, i + \eta\tau]$ . Briefly, for any  $k \in \mathbb{N}$  and  $\delta_b \in (0, 1/2)$  a string  $w \in \{0, 1\}^k$  is a  $\delta_b$ -approximate buffer if  $\text{ED}(w, 0^k) \leq \delta_b \cdot k$ . For brevity we refer to approximate buffers simply as buffers. Once all buffers are found, the algorithm attempts to find a pair of consecutive buffers such that the index  $i$  is between these two buffers. If two such buffers are found, then the algorithm returns these two consecutive buffers. For notational convenience, for integers  $a < b$  we let the tuple  $(a, b)$  denote a (approximate) buffer.

► **Lemma 20.** *Let  $i \in [n']$  and  $j \in [d]$ . There exist constants  $\gamma < \delta_b \in (0, 1/2)$  such that if  $i \in \phi^{-1}(j)$  then **BUFF-FIND** finds buffers  $(a_1, b_1)$  and  $(a_2, b_2)$  such that  $\text{Dec}_{in}(c'[b_1 + 1, a_2 - 1]) = j \circ b_j$ . Further, if  $b_1 < i < a_2$  then **BLOCK-DECODE** outputs  $j \circ b_j$ .*

## 8 Parameter Setting and Proof of Theorem 6

In this section we list a set of constraints which our setting of parameters must satisfy, and then complete the proof of Theorem 6. These constraints are required by different parts of the analysis. Recall that  $\delta_{out}, \delta_{in} \in (0, 1)$  and  $\beta_{in} \geq 1$  are given as parameters of the outer code and the inner code, and that  $\beta = 2\alpha + \beta_{in}(1 + \log |\Sigma|)$ . We have that  $\beta \geq 2$  for any non-negative  $\alpha$ .

► **Proposition 21.** *There exists constants  $\gamma, \theta \in (0, 1)$  and  $\alpha = \Omega(\delta_{in})$  such that the following constraints hold:*

1.  $\gamma \leq 1/12$  and  $\theta < 1/50$ ;
2.  $(\beta + \gamma)/(\beta - \gamma) < 4/3$ ;
3.  $\alpha \leq 2\gamma/(\gamma + 6)$ ;
4.  $\alpha(1 + 3\gamma)/(\beta - 2\alpha) < \delta_{in}$ .

**Proof.** For convenience of the reader and simplicity of the presentation we work with explicit values and verify that they satisfy the constraints in Proposition 21. Let  $\gamma = 1/12$  and  $\theta = 1/51$ , which satisfies constraint (1). Note that  $\gamma < 2/7 \leq \beta/7$ , hence

$$\frac{\beta + \gamma}{\beta - \gamma} < \frac{4}{3}$$

and constraint (2) is satisfied. We take  $\alpha = 2\gamma\delta_{in}/(\gamma + 6)$  so that  $\alpha = \Omega(\delta_{in})$  and constraint (3) is satisfied. Note also that  $\beta - 2\alpha = \beta_{in}(1 + \log |\Sigma|) \geq 2$  which implies

$$\frac{\alpha(1 + 3\gamma)}{\beta - 2\alpha} \leq \frac{\alpha(1 + 3\gamma)}{2} = \frac{\alpha(\gamma + 3\gamma^2)}{2\gamma} < \frac{\alpha(\gamma + 6)}{2\gamma} = \delta_{in}.$$

Therefore, constraint (4) is also satisfied. ◀

We let

$$\delta = \frac{\delta_{out}\alpha\gamma}{2\beta(1 + 1/\theta)} = \Omega(\delta_{in}\delta_{out}).$$

We now prove Theorem 6, which shows Theorem 1.

**Proof of Theorem 6.** Recall that the decoder **Dec** works as follows. Given input index  $i \in [k]$  and oracle access to  $c' \in \{0, 1\}^{n'}$ ,  $\text{Dec}^{c'}(i)$  simulates  $\text{Dec}_{out}^{s'}(i)$ . Whenever  $\text{Dec}_{out}^{s'}(i)$  queries an index  $j \in [m]$ , the decoder expresses  $j = (p - 1)\tau + r_j$  for  $p \in [d]$  and  $0 \leq r_j < \tau$ ,

and runs Algorithm 1 on input  $(c', 1, n' + 1, p)$  to obtain a  $\tau$ -long string  $b'_p$ . Then it feeds the  $(r_j + 1)$ -th symbol of  $b'_p$  to  $\text{Dec}_{out}^{s'}(i)$ . At the end of the simulation,  $\text{Dec}^{c'}(i)$  returns the output of  $\text{Dec}_{out}^{s'}(i)$ .

For  $p \in [d]$ , let  $b'_p \in \Sigma^\tau \cup \{\perp\}$  be a random variable that has the same distribution as the output of Algorithm 1 on input  $(c', 1, n' + 1, p)$ . Define a random string  $s' \in (\Sigma \cup \{\perp\})^m$  as follows. For every  $i \in [m]$  such that  $i = (p - 1)\tau + r$  for  $p \in [d]$  and  $0 \leq r < \tau$ ,

$$s'[i] = \begin{cases} b'_p[r] & \text{if } b'_p \neq \perp, \\ \perp & \text{if } b'_p = \perp. \end{cases}$$

Since  $b'_p = b_p$  implies  $s'[(p - 1)\tau + r] = s[(p - 1)\tau + r]$  for all  $0 \leq r < \tau$ , the event  $E_s := \left\{ \Pr_{j \in [m]} [s'[j] \neq s[j]] \leq \delta_{out} \right\}$  is implied by the event  $E_b := \left\{ \Pr_{j \in [d]} [b'_j \neq b_j] \leq \delta_{out} \right\}$ . Theorem 16 implies that  $\Pr[E_s] \geq \Pr[E_b] \geq 1 - \text{negl}(n')$ . According to the construction of Dec, from the perspective of the outer decoder, the string  $s'$  is precisely the string it is interacting with. Hence by properties of  $\text{Dec}_{out}$  we have that

$$\forall i \in [k], \quad \Pr \left[ \text{Dec}_{out}^{s'}(i) = x[i] \mid E_s \right] \geq \frac{1}{2} + \epsilon_{out}.$$

Therefore by construction of Dec we have

$$\begin{aligned} \forall i \in [k], \quad \Pr \left[ \text{Dec}^{c'}(i) = x[i] \right] &\geq \Pr[E_s] \cdot \Pr \left[ \text{Dec}_{out}^{s'}(i) = x[i] \mid E_s \right] \\ &\geq (1 - \text{negl}(n')) \cdot \left( \frac{1}{2} + \epsilon_{out} \right) \geq \frac{1}{2} + \epsilon_{out} - \text{negl}(n'). \end{aligned}$$

The query complexity of Dec is  $\ell_{out} \cdot O(\log^4 n')$  since it makes  $\ell_{out}$  calls to Algorithm 1, which by Proposition 17 has query complexity  $O(\log^4 n')$ . ◀

---

## References

- 1 Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1001–1017, Dallas, TX, USA, October 31 – November 2 2017. ACM Press. doi:10.1145/3133956.3134031.
- 2 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 99–130, Tel Aviv, Israel, April 29 – May 3 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-78375-8\_4.
- 3 Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Trans. Inf. Theory*, 64(5):3403–3410, 2018.
- 4 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In Shang-Teng Huang, editor, *19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 268–276, San Francisco, CA, USA, January 20–22 2008. ACM-SIAM.
- 5 Aditi Dhagat, Péter Gács, and Peter Winkler. On playing “twenty questions” with a liar. In Greg N. Frederickson, editor, *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 16–22, Orlando, Florida, USA, January 27–29 1992. ACM-SIAM.
- 6 Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM J. Comput.*, 40(4):1154–1178, 2011.

- 7 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.
- 8 Paul Erdős, Ronald L. Graham, and Endre Szemerédi. On sparse graphs with dense long paths, 1975.
- 9 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, October 1994. doi:10.1137/S0097539791195877.
- 10 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recurrent comparison errors, September 2017.
- 11 Venkatesan Guruswami, Bernhard Haeupler, and Amirbehshad Shahrasbi. Optimally resilient codes for list-decoding from insertions and deletions. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 524–537. ACM, 2020.
- 12 Venkatesan Guruswami and Ray Li. Coding against deletions in oblivious and online models. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 625–643. SIAM, 2018.
- 13 Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Transactions on Information Theory*, 65(4):2171–2178, 2018.
- 14 Venkatesan Guruswami and Ray Li. Polynomial time decodable codes for the binary deletion channel. *IEEE Trans. Inf. Theory*, 65(4):2171–2178, 2019.
- 15 Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.
- 16 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 334–347. IEEE Computer Society, 2019.
- 17 Bernhard Haeupler, Aviad Rubinfeld, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and  $(1+\epsilon)$ -approximating edit distance via indexing. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 697–708. ACM, 2019.
- 18 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: codes for insertions and deletions approaching the singleton bound. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 33–46. ACM, 2017.
- 19 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: explicit constructions, local decoding, and applications. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 841–854. ACM, 2018.
- 20 Bernhard Haeupler, Amirbehshad Shahrasbi, and Madhu Sudan. Synchronization strings: List decoding for insertions and deletions. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 21 Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–890, New Orleans, LA, USA, January 7–9 2007. ACM-SIAM.
- 22 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.
- 23 Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. Comput. Syst. Sci.*, 69(3):395–420, 2004.

- 24 Marcos Kiwi, Martin Loeb, and Jiri Matousek. Expected length of the longest common subsequence for large alphabets.
- 25 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant algorithms. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms – ESA 2011*, pages 736–747, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 26 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):11:1–11:42, 2017.
- 27 Swastik Kopparty and Shubhangi Saraf. Guest column: Local testing and decoding of high-rate error-correcting codes. *SIGACT News*, 47(3):46–66, 2016.
- 28 Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *J. ACM*, 61(5):28:1–28:20, 2014.
- 29 Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.
- 30 Shu Liu, Ivan Tjuawinata, and Chaoping Xing. On list decoding of insertion and deletion errors. *CoRR*, abs/1906.09705, 2019. URL: <http://arxiv.org/abs/1906.09705>.
- 31 Hugues Mercier, Vijay K. Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys and Tutorials*, 12, 2010.
- 32 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels, July 2008.
- 33 Rafail Ostrovsky and Anat Paskin-Cherniavsky. Locally decodable codes for edit distance. In Anja Lehmann and Stefan Wolf, editors, *Information Theoretic Security*, pages 236–249, Cham, 2015. Springer International Publishing.
- 34 L. J. Schulman and D. Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, 1999.
- 35 N.J.A. Sloane. On single-deletion-correcting codes. *arXiv: Combinatorics*, 2002.
- 36 Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma (abstract). In *CCC*, page 4, 1999.
- 37 David P. Woodruff. A quadratic lower bound for three-query linear locally decodable codes over any field. *J. Comput. Sci. Technol.*, 27(4):678–686, 2012.
- 38 Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.
- 39 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.



# Maximum Clique in Disk-Like Intersection Graphs

Édouard Bonnet 

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France  
edouard.bonnet@ens-lyon.fr

Nicolas Grelier

Department of Computer Science, ETH Zürich, Switzerland  
nicolas.grelier@inf.ethz.ch

Tillmann Miltzow

Department of Information and Computing Sciences, Utrecht University, The Netherlands  
t.miltzow@googlemail.com

---

## Abstract

We study the complexity of MAXIMUM CLIQUE in intersection graphs of convex objects in the plane. On the algorithmic side, we extend the polynomial-time algorithm for unit disks [Clark '90, Raghavan and Spinrad '03] to translates of any fixed convex set. We also generalize the efficient polynomial-time approximation scheme (EPTAS) and subexponential algorithm for disks [Bonnet et al. '18, Bonamy et al. '18] to homothets of a fixed centrally symmetric convex set.

The main open question on that topic is the complexity of MAXIMUM CLIQUE in disk graphs. It is not known whether this problem is NP-hard. We observe that, so far, all the hardness proofs for MAXIMUM CLIQUE in intersection graph classes  $\mathcal{I}$  follow the same road. They show that, for every graph  $G$  of a large-enough class  $\mathcal{C}$ , the complement of an even subdivision of  $G$  belongs to the intersection class  $\mathcal{I}$ . Then they conclude by invoking the hardness of MAXIMUM INDEPENDENT SET on the class  $\mathcal{C}$ , and the fact that the even subdivision preserves that hardness. However there is a strong evidence that this approach cannot work for disk graphs [Bonnet et al. '18]. We suggest a new approach, based on a problem that we dub MAX INTERVAL PERMUTATION AVOIDANCE, which we prove unlikely to have a subexponential-time approximation scheme. We transfer that hardness to MAXIMUM CLIQUE in intersection graphs of objects which can be either half-planes (or unit disks) or axis-parallel rectangles. That problem is not amenable to the previous approach. We hope that a scaled down (merely NP-hard) variant of MAX INTERVAL PERMUTATION AVOIDANCE could help making progress on the disk case, for instance by showing the NP-hardness for (convex) pseudo-disks.

**2012 ACM Subject Classification** Theory of computation → Computational geometry; Mathematics of computing → Graph algorithms

**Keywords and phrases** Disk Graphs, Intersection Graphs, Maximum Clique, Algorithms, NP-hardness, APX-hardness

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.17

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2003.02583>.

**Funding** *Nicolas Grelier*: Research supported by the Swiss National Science Foundation within the collaborative DACH project Arrangements and Drawings as SNSF Project 200021E-171681.

*Tillmann Miltzow*: NWO Veni Eager 016.Veni.192.250

## 1 Introduction

In an *intersection graph*, the vertices are represented by sets and there is an edge between two sets whenever they intersect. Of course if the sets are not restricted, every graph is an intersection graph. Interesting proper classes of intersection graphs are obtained by restricting the sets to be some specific geometric objects. This comprises unit interval, interval, multiple-interval, chordal, unit disk, disk, axis-parallel rectangle, segment, and string graphs, to name



© Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 17; pp. 17:1–17:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a few. For the most part, they transparently consist of all the intersection graphs of the corresponding objects. Note that *strings* are (polygonal) curves in the plane, and that chordal graphs are the intersection graphs of subtrees in a tree. Intersection graphs have given rise to books (see for instance [39], where applications to biology, psychology, and statistics, are detailed) chapters in monographs (as in [11]), surveys [24, 29], and theses [50]. In this paper we consider objects that are convex sets in the plane.

The MAXIMUM CLIQUE problem on geometric intersection graphs is especially interesting for two reasons. The first reason is that the gap between algorithmic upper and lower bounds is very large, when compared to other usual algorithmic questions on geometric intersection graphs. The second reason is that it has a very natural geometric interpretation, as we will explain below.

We start with a comparison of MAXIMUM CLIQUE with other standard algorithmic problems. The probably most studied problems on geometric intersection graphs are packing and covering problems, for which our theoretical understanding is rather comprehensive. Packing problems (such as MAXIMUM INDEPENDENT SET) and covering problems (such as DOMINATING SET) are often NP-hard in geometric intersection graphs since these problems are already hard on planar graphs. Note for instance that disk intersection graphs [34] and segment intersection graphs [17] both contain all the planar graphs. It turns out that MAXIMUM INDEPENDENT SET (MIS) and DOMINATING SET remain intractable in unit disk, unit square, or segment intersection graphs [37]: Not only are they NP-hard but, being W[1]-hard, they are unlikely to admit a fixed-parameter tractable (FPT), that is,  $f(k)n^{O(1)}$ -time, algorithm, with  $n$  being the input size,  $k$  the size of the solution, and  $f$  any computable function. This intractability is sharply complemented by PTASes for many problems [18, 44, 43, 23, 49, 50], whereas efficient PTASes (EPTASes) are ruled out by the W[1]-hardness of Marx [37]. The existence or unlikelihood of subexponential algorithms for various problems on segment and string graphs was investigated in [10].

On the contrary, many questions are still open when it comes to the computational complexity of MAXIMUM CLIQUE in intersection graphs. Clark et al. [19] show a polynomial-time algorithm for unit disks. A randomized EPTAS, deterministic PTAS, and subexponential-time algorithm were recently published, in top-level conferences [7, 6], for general disk graphs. However neither a polytime algorithm nor NP-hardness is currently known for MAXIMUM CLIQUE on disk graphs. Making progress on this open question is the main motivation of the paper. MAXIMUM CLIQUE was shown NP-hard in segment intersection graphs by Cabello et al. [15]. The proof actually carries over to intersection graphs of unit segments or rays. The existence of an FPT algorithm or of a subexponential-time algorithm for MAXIMUM CLIQUE in segment graphs are both open. MAXIMUM CLIQUE can be solved in polynomial-time in axis-parallel rectangle intersection graphs, since their number of maximal cliques is at most quadratic (every maximal clique corresponds to a distinct cell in any representation). This result was generalized to  $d$ -dimensional polytopes whose facets are all parallel to  $k$  fixed  $(d - 1)$ -dimensional hyperplanes, where MAXIMUM CLIQUE can be solved in time  $n^{O(dk^{d+1})}$  [13]. Note that if the rectangles may have arbitrary slopes, then MAXIMUM CLIQUE is NP-hard since the class then contains segment graphs.

The second reason to study MAXIMUM CLIQUE is that it translates into a very natural question: what is the maximum subset of pairwise intersecting objects? For unit disks, this is equivalent to looking for the maximum subset of centers with (geometric) diameter 2. This is a useful primitive in the context of clustering a given set of points. A related question with a long history is the number of points necessary and sufficient to pierce a collection of pairwise intersecting disks. Danzer [20] and Stacho [48] independently showed that four points

are sufficient and sometimes necessary. Recently Har-Peled et al. [27] gave a linear-time algorithm to find five points piercing a pairwise intersecting collection of disks. A bit later, Carmi et al. [16] obtained a linear-time algorithm for only four points. Note that this implies that we can restrict our attention to instances that can be pierced by 4 points, both for polynomial-time algorithms and hardness reductions.

In this paper, our main focus is the complexity status of `MAXIMUM CLIQUE` on disk graphs. This is a long-standing and seemingly difficult open question. As we will detail in the next paragraphs, there has been basically only one approach to show NP-hardness of `MAXIMUM CLIQUE` in geometric intersection graph classes. This approach is somewhat doomed for the particular case of disk graphs. We develop a new way of showing conditional lower bounds in geometric intersection graphs. We believe that our approach faces a smaller barrier on its way to show that `MAXIMUM CLIQUE` is NP-hard on disk graphs.

### A new alternative to the *co-2-subdivision* approach

`MAXIMUM INDEPENDENT SET` (MIS), which boils down to `MAXIMUM CLIQUE` on the complement graphs, is APX-hard on subcubic graphs [3]. A folklore self-reduction first discovered by Poljak [45] consists of subdividing each edge of the input graph twice (or any even number of times). One can show that this reduction preserves the APX-hardness. Therefore, a way to establish such an intractability for `MAXIMUM CLIQUE` on a given intersection graph class is to show that for every (subcubic) graph  $G$ , the complement of its 2-subdivision  $\text{Subd}_2(G)$  (or  $\text{Subd}_s(G)$  for a larger even integer  $s$ , see [25]) is representable. MIS admits a PTAS on planar graphs, but remains NP-hard. Hence showing that for every (subcubic) planar graph  $G$ , the complement of an even subdivision of  $G$  is representable shows the simple NP-hardness (see [15, 25]).

So far, representing complements of even subdivisions of graphs belonging to a class on which MIS is NP-hard (resp. APX-hard) has been the main, if not unique<sup>1</sup>, approach to show the NP-hardness (resp. APX-hardness) of `MAXIMUM CLIQUE` in geometric intersection graph classes. This approach was used by Middendorf and Pfeiffer [40] for some restriction of string graphs, the so-called  *$B_1$ -VPG graphs*, by Cabello et al. [15] to settle the then long-standing open question of the complexity of `MAXIMUM CLIQUE` for segments (with the class of planar graphs), by Francis et al. [25] for 2-interval, unit 3-interval, 3-track, and unit 4-track graphs (with the class of all graphs; showing APX-hardness), and unit 2-interval and unit 3-track graphs (with the class of planar graphs; showing only NP-hardness), by Bonnet et al. [7] for filled ellipses and filled triangles, and by Bonamy et al. [6] for ball graphs, and 4-dimensional unit ball graphs.

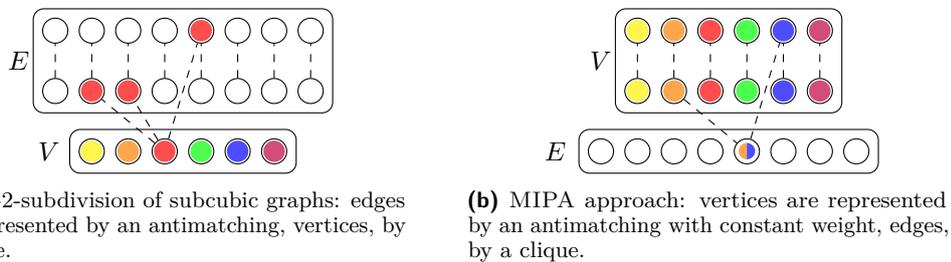
One could hope that the same approach would carry over to show NP-hardness for disk intersection graphs. Bonnet et al. [7] give a structural insight that makes this idea unlikely to work. They showed that the complement of two mutually induced odd cycles is not a disk graph. As a consequence, to show the NP-hardness of `MAXIMUM CLIQUE` on disk graphs with the described approach, one can only hope to represent all the graphs without two mutually induced odd cycles. However we do not know if MIS is even NP-hard in that class. Classifying MIS as NP-hard or polynomial-time solvable on graph classes defined by a list of forbidden induced subgraphs has a long history. Yet the case when all pairs of mutually induced odd cycles are forbidden remains open. Specifically, showing NP-hardness for `MAXIMUM CLIQUE` on disk intersection graphs would resolve this graph-theoretic problem.

<sup>1</sup> Admittedly Butman et al. [14] showed that `MAXIMUM CLIQUE` is NP-hard on 3-interval graphs, by reducing from `MAX 2-DNF-SAT` which is very close to `MAX CUT`. However this result was later subsumed by [25].

The main conceptual contribution of the paper is to suggest an alternative to the standard approach. We introduce an intermediate problem that we call MAX INTERVAL PERMUTATION AVOIDANCE (MIPA, for short), which is a convenient way of seeing MAX CUT. While the definition of MIPA may appear a bit technical (see Section 3), it will give a particularly fitting starting point to design transparent reductions to MAXIMUM CLIQUE in intersection graph classes (as exemplified by Theorem 8).

We prove that MIPA is NP-hard. We then transfer that lower bound to MAXIMUM CLIQUE in the intersection graphs of objects that can be either unit disks or axis-parallel rectangles; a class for which the *co-2-subdivision* approach does not seem to work. Recall that when all the objects are unit disks or when all the objects are axis-parallel rectangles, polynomial-time algorithms are known. The conceptual take-home message is that MIPA can give rise to new geometric hardness results when the *co-2-subdivision* approach fails. For our approach to be applicable to disks as well, it is important that it does not imply APX-hardness, as there is an EPTAS for MAXIMUM CLIQUE on disk graphs [7, 6]. As we even prove that MIPA is APX-hard, there are two options to tackle disk intersection graphs: Either design a reduction which specifically does *not* preserve the inapproximability gap, or restrict MIPA to a simpler NP-hard problem, one admitting a PTAS.

The latter idea of scaling MIPA down can, for instance, be done by replacing the arbitrary matching  $M$  by *pseudo-disk-like* objects, to a NP-hard problem admitting a PTAS. In doing so, one should keep in mind that PLANAR MAX CUT [22] and PLANAR NOT-ALL-EQUAL SAT [41] are solvable in polynomial-time. A next step could be to show the NP-hardness of MAXIMUM CLIQUE for (convex) pseudo-disks. It turns out to be already quite delicate. There is a distinct possibility that convex pseudo-disks have constant induced odd cycle packing number (see Section 2 for the definitions). This would imply a subexponential-time algorithm and an EPTAS [7, 6], and that one would need a scaled down version of MIPA to establish NP-hardness even in that case.



■ **Figure 1** Dashed segments represent non-edges. Both the *co-2-subdivision* and the MIPA approaches require constructing an antimatching and a clique. In the *co-2-subdivision* approach, the *clique vertices* have co-degree 3 to the antimatching. In the MIPA approach their co-degree is only 2. While the difference is seemingly small, the graph class formed by axis-parallel rectangles and unit disks is not amenable to the *co-2-subdivision* approach (see Section 3).

In summary, our main contribution is a new distinct approach to show hardness of MAXIMUM CLIQUE in geometric intersection graphs. Although MIPA is only formally defined in Section 3, one can already see on Figure 1 that both approaches require representing an antimatching (i.e., a complement of an induced matching), a clique, and some relation between them. Antimatchings (and obviously cliques) of arbitrary size are representable by half-planes and unit disks. The difficulty in both cases is to get the right adjacencies between the antimatching and the clique. The MIPA approach only needs the vertices of the clique to *avoid* two vertices in the antimatching, whereas this number is at least three in the *co-2-subdivision* approach. This seemingly small difference is actually crucial, as we will see in Section 3.

## Robustness

Up to this point, we remained vague on how the input intersection graph was given. For, say, disk graphs, do we receive the mere abstract graph or a list of the disks specified by their centers and radii? Computing the graph from the geometric representation can be done efficiently, but not the other way around. Indeed recognizing disk graphs is NP-hard [12] and even  $\exists\mathbb{R}$ -complete [32], where  $\exists\mathbb{R}$  is a class between NP and PSPACE of all the problems polytime reducible to solving polynomial inequalities over the reals. Recognizing string graphs is NP-hard [35], and rather unexpectedly in NP [47], while recognizing segment graphs is  $\exists\mathbb{R}$ -complete [36]. In this context, an algorithm is said to be *robust* if it does not require the geometric representation. A polytime robust algorithm usually decides the problem for a *proper superclass* of the intersection graph class at hand, or correctly reports that the input does not belong to the class. Hence the robust algorithm does not imply an efficient recognition of the class. The polynomial-time algorithm of Clark et al. [19] for MAXIMUM CLIQUE in unit disk intersection graphs requires the geometric representation. Raghavan and Spinrad later extended it to an efficient robust algorithm [46].

## Organization of the paper

The rest of the paper is organized as follows. In Section 2, we introduce the relevant set, graph, and geometry notations and definitions. Then we give the necessary background in hardness of approximation to get ready for the next section. In Section 3, we introduce the MAX INTERVAL PERMUTATION AVOIDANCE problem and prove that it is unlikely to admit a subexponential-time approximation scheme. We use it to show that adding half-planes or unit disks to axis-parallel rectangles is enough for MAXIMUM CLIQUE to go from trivially in P to APX-hard. This is a proof of concept for a different road-map to the *co-even-subdivision* approach, which is compromised for disk graphs. We also observe that if the half-planes are not allowed to be parallel (hence pairwise intersect), then the problem becomes tractable. In Section 4, we extend the EPTAS for disks [7, 6] to homothets of a fixed convex centrally symmetric set. In Section 5, we extend the polytime algorithm for unit disks [19, 46] to translates of a fixed convex set. Our algorithms are robust and our lower bound also holds when the geometric representation is given. Due to lack of space, theorems and lemmas marked with the  $\star$  symbol have their proof deferred to the full version of the paper [8]. Some results had to be moved to the full version entirely.

## 2 Preliminaries

### Sets and graphs

For a pair of positive integers  $i \leq j$ ,  $[i, j]$  denotes the set of all the integers that are at least  $i$  and at most  $j$ , and  $[i]$  is a short-hand for  $[1, i]$ . We overload the notation  $[\cdot, \cdot]$ : If it is explicit or clear from the context that  $x$  or  $y$  is non-integral, then  $[x, y]$  denotes the set of reals that are at least  $x$  and at most  $y$ . We use the usual notations and definitions of graph theory, as they can be found for example in Diestel's book [21]. We denote by  $K_t$ ,  $C_t$ , and  $K_{s,t}$  the complete graph (or clique) on  $t$  vertices, the cycle on  $t$  vertices, and the biclique on  $s$  and  $t$  vertices. The graph  $\bar{G}$  denotes the *complement* of  $G$ , obtained by flipping edges into non-edges, and non-edges into edges. Subdividing an edge  $e = uv$  consists of adding a new vertex linked to both  $u$  and  $v$ , and removing the edge  $e$ . The 2-subdivision  $\text{Subd}_2(G)$  of a graph  $G$  is obtained by subdividing each of its edges twice, hence replacing them by paths of three edges. An even subdivision of a graph  $G$  consists of subdividing every edge of  $G$

an even number of times (potentially zero). A cycle is said to be *induced* if it is chordless. An *odd cycle* (resp. *even cycle*) is a cycle on an odd (resp. even) number of vertices. One can observe that an odd cycle always contains an induced odd cycle. Two cycles are said *mutually induced* if they are chordless and there is no edge linking a vertex of one to a vertex of the other. The induced odd cycle packing number is the maximum number of disjoint odd cycles, that are pairwise mutually induced. An *antimatching* is the complement of an *induced matching* (i.e., a disjoint union of edges). We say that a graph  $G$  is *representable* by some geometric objects, if translates of these objects may have  $G$  as intersection graph.

### Geometric notations

In this paper, we only consider sets in the plane. For two distinct points  $a$  and  $b$ ,  $\ell(a, b)$  denotes the line going through  $a$  and  $b$ . A set  $S$  is *convex* if for any two distinct points  $a$  and  $b$  in  $S$ , the line segment with endpoints  $a$  and  $b$  is contained in  $S$ . It is *bounded*, if it is contained in some disk. A set  $S$  is said to be *centrally symmetric* about the origin if for any point  $a$  in  $S$ ,  $-a$  is also in  $S$ . We mostly deal with sets that are bounded, centrally symmetric, and convex, as they are a natural generalization of disks.

For two sets  $S_1$  and  $S_2$ , we denote by  $S_1 + S_2 := \{s_1 + s_2 \mid s_1 \in S_1, s_2 \in S_2\}$  their Minkowski sum. For the sake of simplicity, for any point  $c$  and any set  $S$ , we denote by  $c + S$  the Minkowski sum of  $\{c\}$  and  $S$ .  $S'$  is a *translate* of  $S$  if there exists  $c$  such that  $S' = c + S$ . Given a positive real number  $\lambda$ ,  $\lambda S$  denotes the set  $\{\lambda s \mid s \in S\}$ . We say that  $S'$  is a *homothet* of  $S$  if there exist a positive real number  $\lambda$  and a point  $c$  such that  $S' = c + \lambda S$ . Moreover we name  $c$  the *center* of  $S'$ , and  $\lambda$  its *scaling factor*.

Let  $F$  be a family of sets in the plane. They form a *pseudo-disk arrangement* if for any pair of sets of  $F$ , their boundaries intersect at most twice. If the sets are also convex we refer to them as *convex pseudo-disks*. They also constitute a natural generalization of disks. A set of rectangles are *axis-parallel* if their boundaries all share the same two slopes. A rectangle is an  $\varepsilon$ -square if its length divided by its width is smaller than  $1 + \varepsilon$ .

### Approximation-schemes

A *polynomial-time approximation scheme* (PTAS) for a maximization problem is an algorithm which takes, together with its input, a parameter  $\varepsilon > 0$  and outputs in time  $n^{f(\varepsilon)}$  a solution of value at least  $(1 - \varepsilon)\text{OPT}$ , where  $\text{OPT}$  is the optimum value. An *efficient PTAS* (EPTAS) is the same but has running time  $f(\varepsilon)n^{O(1)}$ . Note that the existence of an EPTAS, for a problem in which the objective value is the size of the solution  $k$ , implies an FPT algorithm in  $k$ , by setting  $\varepsilon$  to  $1 - \frac{1}{k+1}$ . Indeed in time  $f(1 - \frac{1}{k+1})n^{O(1)} = g(k)n^{O(1)}$ , one then obtains an *exact* solution. A *quasi* PTAS (QPTAS) is an approximation scheme with running time  $n^{\text{polylog } n}$ , for every  $\varepsilon > 0$ . Less standardly, we call *subexponential AS* (SUBEXPAS) an approximation scheme with running time  $2^{n^\gamma}$  for some  $\gamma < 1$ , for every  $\varepsilon > 0$ . These approximation schemes can come deterministic or randomized. A maximization problem  $\Pi$  is *APX-hard* if there is a constant  $\gamma < 1$  such that  $\gamma$ -approximating  $\Pi$  is NP-hard. Unless  $\text{P}=\text{NP}$ , an APX-hard problem cannot admit a PTAS. Ruling out a SUBEXPAS (under admittedly a stronger assumption than  $\text{P}\neq\text{NP}$ ) constitutes a sharper inapproximability than the APX-hardness.

**Strong inapproximability of Positive Not-All-Equal 3-SAT-3.** A longer version of this subsection can be found in the full version of this paper [8]. The Exponential-Time Hypothesis (ETH, for short) of Impagliazzo and Paturi [30] asserts that there is an  $s_3 > 0$  (taking the same

notation as in the original paper) such that 3-SAT cannot be solved in time  $2^{s_3 n}$  on  $n$ -variable instances. By the Sparsification Lemma [31], the ETH implies the same lower bound for 3-SAT-B, in which every variable appears at most a constant  $B$  number of times, depending only on  $s_3$ . Our starting point combines some sharp polytime inapproximability [28], a PCP construction [42], and the Sparsification Lemma.

► **Theorem 1.** [28, 42, 31] *Under the ETH, for every  $\delta > 0$  one cannot distinguish in time  $2^{n^{1-\delta}}$ ,  $n$ -variable  $m$ -clause 3-SAT-instances that are satisfiable from instances where at most  $(7/8 + o(1))m$  clauses can be satisfied, even when each variable appears in at most  $B$  clauses. Thus 3-SAT-B cannot be  $7/8 + o(1)$ -approximated in time  $2^{n^{1-\delta}}$ .*

We recall the definition of NOT-ALL-EQUAL  $k$ -SAT (NAE  $k$ -SAT, for short).

NOT-ALL-EQUAL  $k$ -SAT

**Input:** A conjunction of  $m$  “clauses”  $\phi = \bigwedge_{i \in [m]} C_i$  each on at most  $k$  literals.

**Goal:** Find a truth assignment of the  $n$  variables such that each “clause” has at least one satisfied literal and at least one non-satisfied literal.

The NOT-ALL-EQUAL  $k$ -SAT-B-problem is the same but each variable appears in at most  $B$  clauses (similarly as for  $k$ -SAT-B). The adjective POSITIVE prepended to a satisfiability problem means that no negation (or *negative literal*) can appear in its instances.

► **Theorem 2.** [★] *Under the ETH, for every  $\delta > 0$  one cannot distinguish in time  $2^{n^{1-\delta}}$ ,  $n$ -variable  $m$ -clause NOT-ALL-EQUAL 4-SAT-instances that are satisfiable from instances where at most  $4991m/5000$  clauses can be satisfied, even when each variable appears in at most  $B$  clauses. Thus NOT-ALL-EQUAL 4-SAT-B cannot be  $4991/5000$ -approximated in time  $2^{n^{1-\delta}}$ .*

We now decrease the size of the clauses to at most 3. The next reduction and the subsequent one are folklore. We give complete proofs both for the sake of self-containment and to report explicit inapproximability bounds.

► **Theorem 3.** [★] *Under the ETH, for every  $\delta > 0$  one cannot distinguish in time  $2^{n^{1-\delta}}$ ,  $n$ -variable  $m$ -clause NOT-ALL-EQUAL 3-SAT-instances that are satisfiable from instances where at most  $9991m/10000$  clauses can be satisfied, even when each variable appears in at most  $B$  clauses. Thus NOT-ALL-EQUAL 3-SAT-B cannot be  $9991/10000$ -approximated in time  $2^{n^{1-\delta}}$ .*

Finally, by a linear reduction from NOT-ALL-EQUAL 3-SAT-B to POSITIVE NOT-ALL-EQUAL 3-SAT-3, we decrease the maximum number of occurrences per variable to 3, and we remove the negative literals. A compact yet weaker implication of the following theorem is that a QPTAS for POSITIVE NOT-ALL-EQUAL 3-SAT-3 would disprove the ETH.

► **Theorem 4.** [★] *Under the ETH, for every  $\delta > 0$  one cannot distinguish in time  $2^{n^{1-\delta}}$ ,  $n$ -variable  $m$ -clause POSITIVE NOT-ALL-EQUAL 3-SAT-3-instances that are satisfiable from instances where at most  $\gamma m$  clauses can be satisfied, with  $\gamma := (60000B^2 - 9)/(60000B^2)$ . Thus POSITIVE NOT-ALL-EQUAL 3-SAT-3 cannot be  $\gamma$ -approximated in time  $2^{n^{1-\delta}}$ .*

This last reduction no longer implies APX-hardness. Indeed, the value  $B$  in the inapproximability ratio is finite only if  $s_3 > 0$ . So one should assume the ETH, and not the mere  $P \neq NP$ , to rule out an approximation algorithm with ratio  $\gamma < 1$ . Sacrificing the strong lower bound in the running time, we can overcome that issue. Berman and Karpinski [5] showed that it is NP-hard to approximate MAX 2-SAT-3 within ratio better than  $787/788$ . Following the reduction of Theorem 2 from MAX 2-SAT-3, we derive the following inapproximability.

► **Corollary 5.** *Approximating NAE 3-SAT-10 within ratio  $51326/51327$  is NP-hard.*

**Proof.** Observe that the clause size grows from 2 to 3, and that the variables  $z_j$  are part of at most 9 clauses. ◀

Then following Theorem 4, we get:

► **Corollary 6.** *Approximating POSITIVE NAE 3-SAT-3 within ratio  $49888956/49888957$  is NP-hard.*

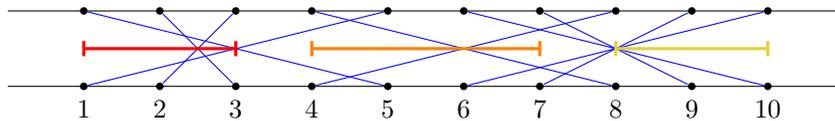
### 3 Max Interval Permutation Avoidance, unit disks and rectangles

We introduce the MAX INTERVAL PERMUTATION AVOIDANCE-problem (MIPA, for short), a convenient intermediate problem to show APX-hardness for geometric problems. We start with an informal description. Let  $M$  be a perfect matching between the  $n$  points  $[n] \times \{0\}$  and  $[n] \times \{1\}$ , in  $\mathbb{N}^2$ . This matching can be represented by a permutation  $\sigma$ , such that for every  $i \in [n]$ ,  $(i, 0)$  is matched with  $(\sigma(i), 1)$ . Imagine now a set of intervals on the line  $y = 1/2$  whose endpoints are all in  $[n]$ . The aim is to move each interval “up” or “down”, by translating it by  $(0, 1/2)$  or by  $(0, -1/2)$ , respectively, such that the number of edges of  $M$  with no endpoint on a translated interval is maximized. An edge of  $M$  with at least one endpoint in a *moved* (or *positioned*) interval is said to be *covered* or *destroyed*. The edge is said to be *uncovered* or *preserved* otherwise. Equivalently MAX INTERVAL PERMUTATION AVOIDANCE aims to minimize the number of covered edges, or maximize the number of uncovered edges. We choose the maximization formulation, since we will both reduce from a maximization problem (POSITIVE NOT-ALL-EQUAL 3-SAT-3) and to a maximization problem (MAXIMUM CLIQUE on disks and rectangles). Thus the objective value will be the number of *uncovered edges*.

MAX INTERVAL PERMUTATION AVOIDANCE

**Input:** A permutation  $\sigma$  over  $[n]$  representing a perfect matching  $M$  between the points  $(1, 0), (2, 0), \dots, (n, 0)$  and  $(\sigma(1), 1), (\sigma(2), 1), \dots, (\sigma(n), 1)$  respectively, and a set of integer ranges  $\mathcal{I} := \{I_1, \dots, I_h\}$  where  $I_k := [\ell_k, r_k]$  and  $1 \leq \ell_k \leq r_k \leq n$ , for every  $k \in [h]$ .

**Goal:** A placement function  $p : \mathcal{I} \rightarrow \{0, 1\}$  encoding that interval  $I_k$  has its endpoints positioned in  $(\ell_k, p(I_k))$  and  $(r_k, p(I_k))$ , which maximizes the number of edges of  $M$  with no endpoint on a positioned interval.



■ **Figure 2** An example of a symmetric instance of MIPA with three disjoint ranges. An optimum solution puts the second interval opposite to the first and third intervals. This leaves 4 edges of the matching uncovered.

A MIPA-input may interchangeably be given as  $(\sigma, \mathcal{I})$  or as  $(M, \mathcal{I})$ . One may observe that a *constant* placement (i.e.,  $p(I_1) = \dots = p(I_h) = 0$ , or  $p(I_1) = \dots = p(I_h) = 1$ ) is a worse solution when the intervals of  $\mathcal{I}$  span  $[n]$ , since it covers all the edges of  $M$ . We say that the matching  $M$  is *symmetric* if  $(i, 0)(j, 1) \in M$  implies that  $(i, 1)(j, 0) \in M$ , for every

$i, j \in [n]$ ; in the geometric viewpoint, it is equivalent to  $y = 1/2$  being a symmetry axis of  $M$ , and in the permutation viewpoint, it is equivalent to  $\sigma$  being a product of pairwise-disjoint transpositions. Other handy (as far as hardness of geometric problems is concerned) technical problems involving intervals and/or permutations include CROSSING-AVOIDING MATCHING in Guśpiel [26] or CROSSING-MINIMIZING PERFECT MATCHING in Guśpiel et al. [2], the problem of covering a 2-track point set by selecting  $k$  2-track intervals [38] or STRUCTURED 2-TRACK HITTING SET [9]. It is no coincidence that these convenient starting problems all involve matchings/permutations and/or intervals. Indeed the latter objects are more easily encoded in a geometric setting than their generalizations: arbitrary binary relations and arbitrary sets. Later we will see how disks can encode intervals and how rectangles can encode a permutation, in the context of the MAXIMUM CLIQUE-problem.

We rule out an approximation scheme for MAX INTERVAL PERMUTATION AVOIDANCE, even if subexponential-time is allowed. In particular a QPTAS for MIPA is highly unlikely. We recall that  $\gamma = (60000B^2 - 9)/(60000B^2)$  and that  $B$  is a finite integral constant, assuming the ETH ( $s_3 > 0$ ).

► **Lemma 7.** *For every  $\delta > 0$ , MAX INTERVAL PERMUTATION AVOIDANCE cannot be  $\gamma'$ -approximated in time  $2^{|M|^{1-\delta}}$ , with  $\gamma' := 1 - (1 - \gamma)/13 < 1$ , unless the ETH fails. Furthermore, MAX INTERVAL PERMUTATION AVOIDANCE is NP-hard and APX-hard. These results hold even if the length of every interval of  $\mathcal{I}$  is at most 5, and the matching  $M$  is symmetric.*

**Proof.** We give a reduction  $\phi$  from POSITIVE NOT-ALL-EQUAL 3-SAT-3 to MAX INTERVAL PERMUTATION AVOIDANCE. Let  $\phi$  be a POSITIVE NAE 3-SAT-3-instance, with variables  $x_1, \dots, x_n$  and clause  $C_1, \dots, C_m$ . For every  $x_i \in C_j$ , we denote by  $\text{occ}(x_i, C_j)$  the number of occurrences of  $x_i$  in the clauses  $C_1, \dots, C_j$ . We observe that  $\text{occ}(x_i, C_j) \in \{1, 2, 3\}$ . We build an instance  $\rho(\phi) := (M, \mathcal{I})$  of MIPA in the following way. For each variable  $x_i$  of  $\phi$ , we reserve a range  $[3(i-1) + 1, 3(i-1) + 3]$  with 3 integral points on both lines  $y = 0$  and  $y = 1$ . These points will be matched by  $M$  to points in the clause gadgets. We add the interval  $X_i := [3(i-1) + 1, 3(i-1) + 3]$  to  $\mathcal{I}$ . We now describe the 2-clause and the 3-clause gadgets.

For every 2-clause  $C_j := x_a \vee x_b$ , we allocate a slot  $S_j$  of size 10 (on  $y = 0$  and  $y = 1$ ) appended to the current last position. The first half of  $S_j$ , that is, the indices in  $[s_j, s_j + 4]$  of  $S_j$  correspond to  $x_a$ , while the indices in  $[s_j + 5, s_j + 9]$  correspond to  $x_b$ . For every  $(d_1, d_2) \in \{(0, 1), (1, 0)\}$  and  $h \in [4]$ , we add to  $M$  the edge between  $(s_j + h, d_1)$  and  $(s_j + 5 + h, d_2)$ . We add the intervals  $C_j(x_a) := [s_j, s_j + 4]$  and  $C_j(x_b) := [s_j + 5, s_j + 9]$  to  $\mathcal{I}$ . Finally for each  $(d_1, d_2) \in \{(0, 1), (1, 0)\}$ , we add to  $M$  the edges between  $(s_j, d_1)$  and  $(3(a-1) + \text{occ}(x_a, C_j), d_2)$ , and between  $(s_j + 5, d_1)$  and  $(3(b-1) + \text{occ}(x_b, C_j), d_2)$ .

For every 3-clause  $C_j := x_a \vee x_b \vee x_c$ , we allocate a slot  $S_j$  of size 15 (on  $y = 0$  and  $y = 1$ ) appended to the current last position. The first third of  $S_j$ , that is, the indices in  $[s_j, s_j + 4]$  of  $S_j$  correspond to  $x_a$ , the second third, the indices in  $[s_j + 5, s_j + 9]$  correspond to  $x_b$ , and the last third, the indices in  $[s_j + 10, s_j + 14]$  correspond to  $x_c$ . We add the intervals  $C_j(x_a) := [s_j, s_j + 4]$ ,  $C_j(x_b) := [s_j + 5, s_j + 9]$ , and  $C_j(x_c) := [s_j + 10, s_j + 14]$  to  $\mathcal{I}$ . Similarly for every  $(d_1, d_2) \in \{(0, 1), (1, 0)\}$  and  $(h, p) \in \{(a, 0), (b, 1), (c, 2)\}$ , we add to  $M$  the edge between  $(s_j + 5p, d_1)$  and  $(3(h-1) + \text{occ}(x_h, C_j), d_2)$ . We call these edges *internal* (same for the 2-clause gadget). Finally we add to  $M$  four edges from every pair of ranges in  $\{[s_j, s_j + 4], [s_j + 5, s_j + 9], [s_j + 10, s_j + 14]\}$ , two starting on the line  $y = 0$  (ending on  $y = 1$ ) and two starting on  $y = 1$  (ending on  $y = 0$ ). We call these edges *variable-clause* (same for the 2-clause gadget).

For each variable  $x_i$  with only two occurrences in  $\phi$ , we link its third occurrence pair to a dummy pair  $(d_i, 0), (d_i, 1)$ , appended to the current last position. That is, we add the edges  $(3(i-1) + 3, 0)(d_i, 1)$  and  $(3(i-1) + 3, 1)(d_i, 0)$  to  $M$ . Although not needed, we also

## 17:10 Maximum Clique in Disk-Like Intersection Graphs

add the singleton interval  $D_i := \{d_i\}$  to  $\mathcal{I}$ . We call it *dummy gadget* and consider it as a special case of a clause gadget. This finishes the construction of the MIPA-instance  $(M, \mathcal{I})$ . Observe that every point is matched, and that all the intervals of  $\mathcal{I}$  are pairwise disjoint, and of length at most 5. The perfect matching  $M$  comprises at most  $3n + 15m + n \leq 49n$  edges.

We assume that  $\phi$  is satisfiable, and let  $\mathcal{V}$  be a satisfying assignment. We build the following solution to the MIPA-instance. We push the interval  $X_i$  up (to the line  $y = 1$ ) if  $x_i$  is set to true by  $\mathcal{V}$ , and we push it down (to the line  $y = 0$ ) otherwise. In the clause gadgets (and dummy gadgets), we do the opposite: we push  $C_j(x_i)$  ( $D_i$ ) down if  $x_i$  is set to true, and up if  $x_i$  is set to false. This solution preserves four edges within each clause gadget, and an additional  $3n$  edges between the variable gadgets and the clause gadgets. Hence the total number of preserved edges is  $4m + 3n$ .

We now assume that at most  $\gamma m$  clauses of  $\phi$  are satisfiable. Let  $p$  be a placement function of the intervals of  $\mathcal{I}$ , maximizing the number of preserved edges of  $M$ . We first argue that not giving the same placement (up/1 or down/0) to the three (resp. two) intervals  $C_j(x_a), C_j(x_b), C_j(x_c)$  (resp.  $C_j(x_a), C_j(x_b)$ ) of a 3-clause gadget (resp. 2-clause gadget) is always better. Note that any equal placement destroys all the edges of  $M$  internal to the clause gadget of  $C_j$ , and preserves at most three variable-clause edges. On the other hand, a placement with at least one interval on each side preserves already four internal edges. We can then assume that  $p$  does not give equal placement in any clause gadget. Let  $\mathcal{V}$  be the assignment of the variables of  $\phi$  which sets  $x_i$  to true if  $p(X_i) = 1$ , and to false, if  $p(X_i) = 0$ . By assumption  $\mathcal{V}$  does not satisfy at least  $(1 - \gamma)m$  clauses. In each corresponding clause gadget, one can preserve at most two variable-clause edges of  $M$ . Indeed, since  $\phi$  is a POSITIVE NAE 3-SAT-3-instance, all three variable-clause edges incident to the clause gadget and not covered by the placement of the  $X_i$  land on the same side. By the previous remark, at least one such edge should be destroyed (to preserve four internal edges). Thus the placement  $p$  preserves at most  $3n + 4m - (1 - \gamma)m$  edges.

Since  $|M| = O(n + m) = O(n)$  and  $\frac{3n + 4m - (1 - \gamma)m}{3n + 4m} \leq 1 - \frac{1 - \gamma}{13}$ , by Theorem 4 MIPA cannot be  $\gamma'$ -approximated in time  $2^{|M|^{1 - \delta}}$ , under the ETH. Besides, by Corollary 6, MIPA cannot be  $648556435/648556436$ -approximated in polynomial-time, unless  $P = NP$ . In particular, this problem is NP-hard and even APX-hard.  $\blacktriangleleft$

We recall that MAXIMUM CLIQUE can be solved in polynomial-time in unit disk graphs [19, 46] and in axis-parallel rectangle intersection graphs [13]. Now if the objects can be unit disks *and* axis-parallel rectangles, we show that even a SUBEXPAS is unlikely. We denote by  $\{\text{OBJ}, \text{OBJ}'\}$ -MAXIMUM CLIQUE the clique problem in the intersection graphs of objects that can be either OBJ or OBJ'.

► **Theorem 8.** *For every  $\delta > 0$ , MAXIMUM CLIQUE in intersection graphs  $G$  of unit disks and axis-parallel rectangles cannot be  $c$ -approximated in time  $2^{|V(G)|^{1 - \delta}}$ , with  $c := 1 - (1 - \gamma)/153 < 1$ , unless the ETH fails. Moreover, this problem is NP-hard and APX-hard.*

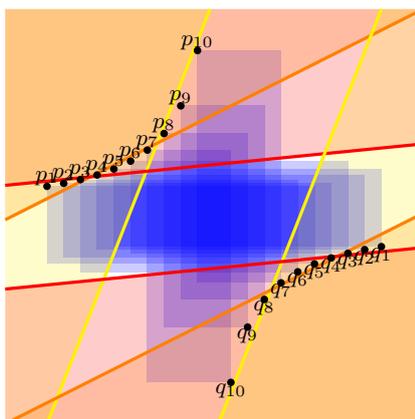
**Proof.** We give a reduction from MAX INTERVAL PERMUTATION AVOIDANCE to  $\{\text{UNIT DISKS}, \text{AXIS-PARALLEL RECTANGLES}\}$ -MAXIMUM CLIQUE or  $\{\text{HALF-PLANES}, \text{AXIS-PARALLEL RECTANGLES}\}$ -MAXIMUM CLIQUE. Let  $(M, \mathcal{I})$  be an instance of MIPA over  $[n]$ , where  $M$  is symmetric, and all the intervals of  $\mathcal{I}$  have size at most 5. We build the following set of axis-parallel rectangles  $\mathcal{R}$  and half-planes  $\mathcal{H}$ . See Figure 3 for an illustration.

Let  $O$  be the origin of the plane. We place from left to right  $n + 2$  points  $p_0, p_1, \dots, p_n, p_{n+1}$  on a convex curve in the top-left quadrant, say  $x \mapsto -1/x$  on  $[-(1 + \lambda), -1]$  for some small  $\lambda > 0$ . We wiggle the points  $p_i$  so that for every  $i \leq j \in [n]$ , the slope of the line passing through  $\text{middle}(p_{i-1}, p_i)$  and  $\text{middle}(p_j, p_{j+1})$  has a distinct value, where  $\text{middle}(p, q)$  denotes

the midpoint of the segment with endpoints  $p$  and  $q$ . We define  $q_0, q_1, \dots, q_n, q_{n+1}$ , such that  $O$  is the middle of the segment  $p_i q_i$  for every  $i \in [0, n + 1]$ . In other words, this new chain is obtained by central symmetry about  $O$ . Observe that sorted by  $x$ -coordinates, these  $2n + 4$  points read  $p_0, p_1, \dots, p_n, p_{n+1}, q_{n+1}, q_n, \dots, q_1, q_0$ . The points  $p_1, \dots, p_n$  represent  $[n] \times \{0\}$  in the MIPA-instance, while the points  $q_1, \dots, q_n$  represent  $[n] \times \{1\}$ .

For every pair  $i \leq j \in [n]$ , we can associate a line  $\ell_p(i, j)$  passing through  $\text{middle}(p_{i-1}, p_i)$  and  $\text{middle}(p_j, p_{j+1})$ . Notice that, by convexity,  $\ell_p(i, j)$  separates the points  $p_i, p_{i+1}, \dots, p_{j-1}, p_j$  (below it) from the points  $p_1, \dots, p_{i-1}, p_{j+1}, \dots, p_n$  (above it). We similarly define  $\ell_q(i, j)$  as the line passing through  $\text{middle}(q_{i-1}, q_i)$  and  $\text{middle}(q_j, q_{j+1})$ . We observe that  $\ell_p(i, j)$  and  $\ell_q(i, j)$  are parallel. For every interval  $I = [i, j] \in \mathcal{I}$ , we introduce in the MAXIMUM CLIQUE-instance the half-plane  $h_p(I) := h_p(i, j)$  as the closed upper half-plane whose boundary is  $\ell_p(i, j)$ , and  $h_q(I) := h_q(i, j)$  as the closed lower half-plane whose boundary is  $\ell_q(i, j)$ . We give these two objects weight 5 by superimposing 5 copies of them. All pairs of introduced half-planes intersect, except the pairs  $\{h_p(i, j), h_q(i, j)\}$ .

Finally for every edge  $(i, 0)(j, 1)$  of the matching  $M$  (with  $i, j \in [n]$ ), we add an axis-parallel rectangle  $R(i, j)$  whose top-left corner is  $p_i$  and bottom-right corner is  $q_j$ . This finishes the construction of  $(\mathcal{R}, \mathcal{H})$ . When  $\lambda$  tends to 0, the rectangles are arbitrary close to squares of equal side-length. In other words, for any  $\varepsilon > 0$ , the axis-parallel rectangles can be made  $\varepsilon$ -squares. The half-planes can be turned into unit disks, making the side-length of the rectangles very small compared to 1. We denote by  $(\mathcal{R}, \mathcal{D})$  the corresponding sets of axis-parallel rectangles and unit disks, and by  $G$  their intersection graph.



■ **Figure 3** The output of the reduction on the instance of Figure 2.

Let us consider instances of MIPA produced by the previous reduction from POSITIVE NAE 3-SAT-3, on  $\nu$ -variable  $\mu$ -clause formulas that are either satisfiable or with at least  $(1 - \gamma)\mu$  non satisfiable clauses. We call *yes-instances* the former MIPA-instances, and *no-instances*, the latter. If  $(M, \mathcal{I})$  is a yes-instance, we claim that  $G$  has a clique of size  $5|\mathcal{I}| + 3\nu + 4\mu$ . Indeed there is a placement  $p$  that preserves  $3\nu + 4\mu$  edges of  $M$ . We start by taking in the clique all the half-planes (or corresponding unit disks)  $h_p(I)$  whenever  $p(I) = 0$ , and  $h_q(I)$  whenever  $p(I) = 1$ . Since these objects have weight 5 (actually 5 stacked copies), this amounts to  $5|\mathcal{I}|$  vertices. The corresponding half-planes pairwise intersect since their boundaries have distinct slopes. Then we include to the clique the  $3\nu + 4\mu$  rectangles  $R(i, j) \in \mathcal{R}$  such that  $(i, 0)(j, 1)$  is preserved by  $p$ . All the rectangles pairwise intersect since they all contain the origin  $O$ . Every pair of chosen half-plane  $h_z(I)$  ( $z \in \{p, q\}$ ) and rectangle  $R(a, b)$  intersects, otherwise the placement of  $I$  would cover  $(a, 0)(b, 1)$ . Thus we exhibited a clique of size  $5|\mathcal{I}| + 3\nu + 4\mu$  in  $G$ .

## 17:12 Maximum Clique in Disk-Like Intersection Graphs

We now assume that  $(M, \mathcal{I})$  is a no-instance, and we claim that  $G$  has no clique larger than  $5|\mathcal{I}| + 3\nu + 4\mu - (1 - \gamma)\mu$ . Let us see how to build a clique in  $G$ . One can take at most one object between  $h_p(I)$  and  $h_q(I)$  (since they do not intersect). There is a maximum clique that takes at least one of  $h_p(I)$  and  $h_q(I)$  since  $h_p(I)$  has weight 5 and intersects every object but  $h_q(I)$  plus at most 5 rectangles (recall that the intervals of  $\mathcal{I}$  have size at most 5). Thus we assume that our maximum clique takes exactly one object between  $h_p(I)$  and  $h_q(I)$ , for every  $I \in \mathcal{I}$ . We consider the placement  $p$  defined as  $p(I) = 0$  if  $h_p(I)$  is in the clique, and  $p(I) = 1$  if  $h_q(I)$  is in the clique. Now the rectangles  $R(i, j)$  that are adjacent to the chosen half-planes of  $\mathcal{H}$  (or unit disks of  $\mathcal{D}$ ) correspond to the edges  $(i, 0)(j, 1)$  of  $M$  which are preserved. By Lemma 7, there are at most  $3\nu + 4\mu - (1 - \gamma)\mu$  such rectangles.

Since  $|V(G)| = |\mathcal{H}| + |\mathcal{R}| = 10|\mathcal{I}| + |M| = O(\nu + \mu) = O(\nu)$  and  $\frac{5|\mathcal{I}| + 3\nu + 4\mu - (1 - \gamma)\mu}{5|\mathcal{I}| + 3\nu + 4\mu} \leq 1 - \frac{(1 - \gamma)\mu}{140\mu + 9\nu + 4\mu} = 1 - \frac{1 - \gamma}{153} = c$ , by Theorem 4, {HALF-PLANES/UNIT DISKS, AXIS-PARALLEL RECTANGLES}-MAXIMUM CLIQUE cannot be  $c$ -approximated in time  $2^{|V(G)|^{1-\delta}}$ , under the ETH. Additionally, by Corollary 6, this problem cannot be 7633010347/7633010348-approximated in polynomial-time, unless P=NP. In particular, it is NP-hard and even APX-hard. ◀

We observe that if all the half-planes pairwise intersect (for instance because their boundaries are assumed to have distinct slopes), then there is a polynomial-time algorithm, given a geometric representation. Let again  $\mathcal{H}$  be the half-planes and  $\mathcal{R}$ , the axis-parallel rectangles, in the representation of the graph  $G$ . Recall that the number of maximal cliques in  $G[\mathcal{R}]$  is polynomial, and that they can be enumerated efficiently. For each maximal clique  $\mathcal{R}_c \subseteq \mathcal{R}$ , we compute the maximum clique in the co-bipartite graph  $G[\mathcal{H} \cup \mathcal{R}_c]$ . This is thus equivalent to computing MIS in a bipartite graph. Due to König's theorem, this can be done in polynomial-time by a matching algorithm. We output  $C$  the largest clique that we find.  $C$  is a maximum clique in  $G$ , since  $C \cap \mathcal{R}$  is by definition a clique, so it is contained in a maximal clique of  $G[\mathcal{R}]$ .

Let us briefly discuss the issue the *co-2-subdivision* approach encounters for {HALF-PLANES, AXIS-PARALLEL RECTANGLES}-MAXIMUM CLIQUE. Axis-parallel rectangles cannot represent a large antimatching (they already cannot represent  $\overline{3K_2}$ ). Hence, as in our construction, the large antimatching has to be, for the most part, realized by half-planes. Now in the MIPA approach, the axis-parallel rectangles can avoid *two* arbitrary half-planes with the freedom of their top-left and bottom-right corners. In the *co-2-subdivision* approach, they would have to avoid *at least three* arbitrary half-planes, and do not have enough degrees of freedom for that.

### 4 Homothets of a centrally symmetric convex set

Here we observe that the EPTAS for MAXIMUM CLIQUE in disk graphs extends to the intersection graphs of homothets of a centrally symmetric convex set. Bonamy et al. show:

► **Theorem 9** ([6]). *For any constants  $d \in \mathbb{N}$ ,  $0 < \beta \leq 1$ , for every  $0 < \varepsilon < 1$ , there is a randomized  $(1 - \varepsilon)$ -approximation algorithm running in time  $2^{\tilde{O}(1/\varepsilon^3)} n^{O(1)}$ , and a deterministic PTAS running in time  $n^{\tilde{O}(1/\varepsilon^3)}$  for MAXIMUM CLIQUE on  $n$ -vertex graphs  $G$  satisfying the following conditions:*

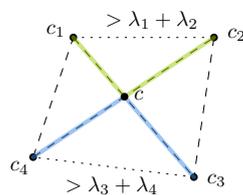
- *there are no two mutually induced odd cycles in  $\overline{G}$  (the complement of  $G$ ),*
- *the VC-dimension of the neighborhood hypergraph  $\{N[v] \mid v \in V(G)\}$  is at most  $d$ , and*
- *$G$  has a clique of size at least  $\beta n$ .*

The first item is enough to obtain a subexponential-algorithm [7] and boils down to proving a structural lemma on the representation of  $K_{2,2}$  (see Lemma 11). We show that the previous theorem applies to more general shapes than disks.

► **Theorem 10.** *MAXIMUM CLIQUE admits a subexponential-time algorithm and an EPTAS in intersection graphs of homothets of a fixed bounded centrally symmetric convex set  $S$ .*

Let  $S$  be a centrally symmetric, bounded, convex set. We can define a corresponding norm as follow: for any  $x \in \mathbb{R}^2$ , let  $\|x\|$  be equal to  $\inf\{\lambda > 0 \mid x \in \lambda S\}$ . This is well-defined since  $S$  is bounded. It is absolutely homogeneous because  $S$  is centrally symmetric, and it is subadditive because  $S$  is convex. Therefore  $\|\cdot\|$  is a norm. We use the norm we have defined, and check the three conditions of Theorem 9.

► **Lemma 11.** *In a representation of  $K_{2,2}$  with homothets of  $S$  placing the four centers in convex position, the non-edges are between vertices corresponding to opposite corners of the quadrangle.*



■ **Figure 4** Illustration of the proof of Lemma 11. Non-edges are dotted and edges are dashed.

**Proof.** Let  $S_1, S_2, S_3$  and  $S_4$  be the four homothets. We denote by  $c_i$  the center of  $S_i$ , and by  $\lambda_i$  its scaling factor. Let us assume by contradiction that they appear in this order on the convex hull, that  $S_1$  and  $S_2$  make one non-edge, and that  $S_3$  and  $S_4$  make the other. By assumption, we have  $\|c_1 - c_2\| > \lambda_1 + \lambda_2$ , and likewise  $\|c_3 - c_4\| > \lambda_3 + \lambda_4$ . Let us denote by  $c$  the intersection of the lines  $\ell(c_1, c_3)$  and  $\ell(c_2, c_4)$ , where  $\ell(p, q)$  denotes the line going through two distinct points  $p$  and  $q$ . We have  $\|c_1 - c\| + \|c - c_2\| > \|c_1 - c_2\|$  by triangular inequality. Likewise it holds  $\|c_3 - c\| + \|c - c_4\| > \|c_3 - c_4\|$ . We therefore obtain  $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 < \|c_1 - c\| + \|c - c_2\| + \|c_3 - c\| + \|c - c_4\| = \|c_1 - c_3\| + \|c_2 - c_4\| \leq \lambda_1 + \lambda_3 + \lambda_2 + \lambda_4$ , which is a contradiction. ◀

Lemma 11 implies by some parity arguments that the first condition of Theorem 9 holds (see Theorem 6 in [7]). It is well known that a family of homothets forms a pseudo-disk arrangement. Therefore the second property holds as shown by Aronov et al. [4]. Finally we enforce the third condition of Theorem 9, by using a chi-boundedness result of Kim et al. [33].

► **Lemma 12.** *With a polynomial multiplicative factor in the running time, one can reduce to instances satisfying the third condition of Theorem 9 with  $\beta = 1/36$ .*

**Proof.** Kim et al. [33] show that in any representation of an intersection graph  $G$  of homothets of a convex set, a homothet  $S$  with a smallest scaling factor has degree at most  $6\omega(G) - 7$ , where  $\omega(G)$  denotes the clique number of  $G$ . Their proof also implies that the independence number of its neighborhood is at most 6. By degenerence, the coloring number, denoted by  $\chi(G)$  is at most  $6\omega(G) - 6$ . First we find in polynomial-time a vertex  $v$  such that the independence number of its neighborhood is at most 6. Let us denote by  $G_v$  the subgraph induced by its neighborhood, and  $n$  denotes its number of vertices. We denote by  $\alpha(\cdot)$  the

## 17:14 Maximum Clique in Disk-Like Intersection Graphs

independence number of a graph. As  $G_v$  has a representation with homothets of  $S$ , we have  $\chi(G_v) \leq 6\omega(G_v)$ . Therefore  $\alpha(G_v)\omega(G_v) \geq \frac{1}{6}\alpha(G_v)\chi(G_v) \geq \frac{1}{6}n$ . Thus by assumption we have  $\omega(G_v) \geq \frac{1}{36}n$ . Then we can compute a maximum clique that contains  $v$ , or remove  $v$  from the graph and iterate. The EPTAS of Bonamy et al. is called linearly many times. ◀

### 5 Translates of a convex set

We show in this section that we can extend the algorithm of Clark et al. [19] and its robust version [46] from unit disks to any centrally symmetric, bounded, convex set.

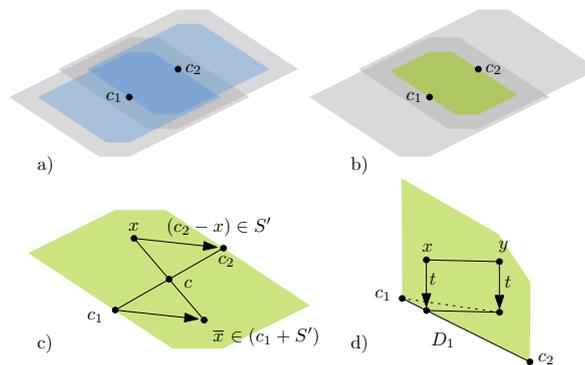
► **Theorem 13.** *MAXIMUM CLIQUE admits a robust polynomial-time algorithm in intersection graphs of translates of a fixed centrally symmetric, bounded, convex set.*

Moreover, as shown by Aamand et al. [1], for every bounded and convex set  $S_1$ , there exists a centrally symmetric, bounded and convex set  $S_2$  such that  $\mathcal{G}_{S_1} = \mathcal{G}_{S_2}$ , where  $\mathcal{G}_S$  denotes the intersection graphs class of translates of  $S$ . Thus we obtain the immediate corollary:

► **Corollary 14.** *MAXIMUM CLIQUE admits a robust polynomial-time algorithm in intersection graphs of translates of a fixed bounded and convex set.*

We prove Theorem 13 in two steps. First we show how to compute in polynomial time a maximum clique when a representation is given. Secondly we use the result by Raghavan and Spinrad [46] to obtain a robust algorithm.

Let  $S$  be a centrally symmetric, bounded, convex set. We use the norm defined in Section 4: for any  $x \in \mathbb{R}^2$ , let  $\|x\|$  be equal to  $\inf\{\lambda > 0 \mid x \in \lambda S\}$ . Let  $S_1$  and  $S_2$  be two translates of  $S$ , with respective centers  $c_1$  and  $c_2$ . Remark that  $S_1$  and  $S_2$  intersect if and only if  $\|c_1 - c_2\| \leq 2$ . Let us assume that  $d := \|c_1 - c_2\| \leq 2$ . We denote by  $S'$  the set  $S$  scaled by  $d$ :  $S' := dS$ , and we then define:  $D := \{x \in \mathbb{R}^2 \mid \|x - c_1\| \leq d, \|x - c_2\| \leq d\}$ . Equivalently we have  $D = (c_1 + S') \cap (c_2 + S')$ . If  $S$  was a unit disk,  $D$  would be the intersection of two disks with radius  $d$ , such that the boundary of one contains the center of the other.



► **Figure 5** a) The gray sets are scaled about their center so that the center of one set is on the boundary of the other. b) the intersection  $D$ . c) Illustration of Lemma 15. d) Illustration of Lemma 17.

► **Lemma 15.** *The set  $D$  is centrally symmetric around  $c := (c_1 + c_2)/2$ .*

**Proof.** Let  $x$  be a point in  $D$ , we need to show that  $\bar{x} := x + 2(c - x)$  is in  $D$  too. As  $D = (c_1 + S') \cap (c_2 + S')$ , it is sufficient to show  $\bar{x} \in c_1 + S'$  and  $\bar{x} \in c_2 + S'$ . By definition,  $\bar{x}$  is equal to  $c_1 + c_2 - x$ . Since  $x$  is in  $D$ , then  $\|c_2 - x\| \leq d$ , which implies that  $c_2 - x$  is in  $S'$ . Therefore  $\bar{x}$  is in  $c_1 + S'$ . By the symmetry of the arguments, we obtain that  $\bar{x}$  is in  $D$ . ◀

► **Lemma 16.** *The tangents to  $D$  at  $c_1$  and  $c_2$  are parallel.*

**Proof.** Let us denote by  $\ell_1$  the tangent to  $D$  at  $c_1$ . Then we denote by  $\ell_2$  the line parallel to  $\ell_1$  that contains  $c_2$ . We claim that  $\ell_2$  is tangent to  $D$ . By construction  $D$  is convex, as the intersection of two convex sets. This implies that  $\ell_2$  is tangent to  $D$  if and only if  $D \cap \ell_2$  is a line segment that contains  $c_2$ . This line segment may be only one point. Let  $x$  be a point in  $D \cap \ell_2$ . By Lemma 15,  $D$  is centrally symmetric around  $c$ . Therefore  $x + 2(c - x)$  is in  $D$ , and by construction it is also in  $\ell_1$ . Since  $D \cap \ell_1$  is a line segment that contains  $c_1$ , thus  $D \cap \ell_2$  is a line segment that contains  $c_2$ . ◀

We cut  $D$  along the line  $\ell$  going through  $c_1$  and  $c_2$ , and split  $D$  into two sets denoted by  $D_1$  and  $D_2$ . We define  $D_1$  as the set of points below this line, and  $D_2$  as the set of points not below.

► **Lemma 17.** *Let  $i$  be in  $\{1, 2\}$ , and let  $x$  and  $y$  be in  $D_i$ . Then we have  $\|x - y\| \leq d$ .*

**Proof.** We do the proof for  $i = 1$ , and the case  $i = 2$  can be done symmetrically. By Lemma 16, the tangents  $\ell_1$  and  $\ell_2$  of  $D$  at  $c_1$  and  $c_2$  are parallel. Without loss of generality, let us assume that they are vertical, that  $c_1$  is to the left of  $c_2$  and  $x$  to the left of  $y$ . We denote by  $\tilde{x}$  (respectively  $\tilde{y}$ ) the vertical projection of  $x$  (respectively  $y$ ) on  $\ell$ . Without loss of generality  $\|x - \tilde{x}\| \leq \|y - \tilde{y}\|$ . We define  $t = x - \tilde{x}$ . Note that  $\|x - y\| = \|(x - t) - (y - t)\| = \|\tilde{x} - (y - t)\|$ . Furthermore, we can move  $\tilde{x}$  on  $\ell$  towards  $c_1$  and this will only increase the distance to  $(y - t)$ . We get  $\|\tilde{x} - (y - t)\| \leq \|c_1 - (y - t)\|$ . By definition  $(y - t) \in D_1 \subset D$  and thus  $\|c_1 - (y - t)\| \leq d$ . This implies  $\|x - y\| \leq d$  and finishes the proof. ◀

Following the arguments of Clark et al. [19], one first guesses in quadratic time  $S_1$  and  $S_2$  in a maximum clique  $C$  such that the distance between their centers  $\|c_1 - c_2\|$  is maximized among the pairs  $S_1, S_2 \in C$ . One can then remove all the objects not centered in  $D$ . By Lemma 17, the intersection graph induced by the sets centered in  $D$  is cobipartite. Since computing an independent set in a bipartite graph can be done in polynomial time, then one can compute a maximum clique in  $G$  in polynomial time.

Before explaining how to compute a maximum clique when no representation is given, we need to introduce a few definitions. Let  $\Lambda = e_1, e_2, \dots, e_m$  be an ordering of the  $m$  edges of  $G$ . Let  $G_\Lambda(k)$  be the subgraph of  $G$  with edge-set  $\{e_k, e_{k+1}, \dots, e_m\}$ . For each  $e_k = uv$ ,  $N_{\Lambda,k}$  is defined as the set of vertices adjacent to  $u$  and  $v$  in  $G_\Lambda(k)$ .

► **Definition 18** (Raghavan and Spinrad [46]). An edge ordering  $\Lambda = e_1, e_2, \dots, e_m$  is a *cobipartite neighborhood edge elimination ordering* (CNEEO), if for each  $e_k$ ,  $N_{\Lambda,k}$  induces a cobipartite graph in  $G$ .

**Proof of Theorem 13.** Raghavan and Spinrad have given a polynomial time algorithm that takes an abstract graph as input, and returns a CNEEO or a certificate that no CNEEO exists for the graph. Secondly, they showed how to compute in polynomial time a maximum clique when given a graph and a CNEEO on it. Therefore, it is sufficient to show that for any centrally symmetric, bounded, convex set  $S$ , and any intersection graph  $G$  of translated of  $S$ , there exists a CNEEO on  $G$ . Let us consider such a graph  $G$  with a representation. Arguing with Lemma 17 as previously, ordering the edges by non-increasing length gives a CNEEO, where the length of an edge is the distance between the two centers. ◀

## References

- 1 Anders Aamand, Mikkel Abrahamsen, Jakob B. T. Knudsen, and Peter M. R. Rasmussen. Classifying convex bodies by their contact and intersection graphs. *CoRR*, abs/1902.01732, 2019. [arXiv:1902.01732](https://arxiv.org/abs/1902.01732).
- 2 Akanksha Agrawal, Grzegorz Guspiel, Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Connecting the dots (with minimum crossings). In *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA.*, pages 7:1–7:17, 2019. doi:10.4230/LIPIcs.SoCG.2019.7.
- 3 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.
- 4 Boris Aronov, Anirudh Donakonda, Esther Ezra, and Rom Pinchasi. On pseudo-disk hypergraphs. *arXiv preprint arXiv:1802.08799*, 2018.
- 5 Piotr Berman and Marek Karpinski. Efficient amplifiers and bounded degree optimization. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(53), 2001. URL: <http://eccc.hpi-web.de/eccc-reports/2001/TR01-053/index.html>.
- 6 Marthe Bonamy, Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, and Stéphan Thomassé. EPTAS for max clique on disks and unit balls. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 568–579, 2018. doi:10.1109/FOCS.2018.00060.
- 7 Édouard Bonnet, Panos Giannopoulos, Eun Jung Kim, Paweł Rzażewski, and Florian Sikora. QPTAS and subexponential algorithm for maximum clique on disk graphs. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pages 12:1–12:15, 2018. doi:10.4230/LIPIcs.SoCG.2018.12.
- 8 Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. *arXiv preprint arXiv:2003.02583*, 2020.
- 9 Édouard Bonnet and Tillmann Miltzow. Parameterized hardness of art gallery problems. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 19:1–19:17, 2016. doi:10.4230/LIPIcs.ESA.2016.19.
- 10 Édouard Bonnet and Paweł Rzażewski. Optimality program in segment and string graphs. *Algorithmica*, 81(7):3047–3073, 2019. doi:10.1007/s00453-019-00568-7.
- 11 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM, 1999.
- 12 Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2):3–24, 1998. doi:10.1016/S0925-7721(97)00014-X.
- 13 Valentin E. Brimkov, Konstanty Junosza-Szaniawski, Sean Kafer, Jan Kratochvíl, Martin Pergel, Paweł Rzażewski, Matthew Szczepankiewicz, and Joshua Terhaar. Homothetic polygons and beyond: Maximal cliques in intersection graphs. *Discrete Applied Mathematics*, 247:263–277, 2018. doi:10.1016/j.dam.2018.03.046.
- 14 Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6(2):40:1–40:18, 2010. doi:10.1145/1721837.1721856.
- 15 Sergio Cabello, Jean Cardinal, and Stefan Langerman. The clique problem in ray intersection graphs. *Discrete & Computational Geometry*, 50(3):771–783, 2013. doi:10.1007/s00454-013-9538-5.
- 16 Paz Carmi, Matthew J. Katz, and Pat Morin. Stabbing pairwise intersecting disks by four points. *CoRR*, abs/1812.06907, 2018. [arXiv:1812.06907](https://arxiv.org/abs/1812.06907).
- 17 Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 631–638, 2009. doi:10.1145/1536414.1536500.
- 18 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.

- 19 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-0.
- 20 Ludwig Danzer. Zur lösung des gallaischen problems über kreisscheiben in der euklidischen ebene. *Studia Sci. Math. Hungar.*, 21(1-2):111–134, 1986.
- 21 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 22 Y. G. Dorfman and G. I. Orlova. Finding the maximal cut in a graph. *Engineering Cybernetics*, 10:502–506, 1972.
- 23 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.
- 24 Aleksei V. Fishkin. Disk graphs: A short survey. In *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, pages 260–264, 2003. doi:10.1007/978-3-540-24592-6\_23.
- 25 Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The Maximum Clique Problem in Multiple Interval Graphs. *Algorithmica*, 71(4):812–836, 2015. doi:10.1007/s00453-013-9828-6.
- 26 Grzegorz Guspriel. Complexity of finding perfect bipartite matchings minimizing the number of intersecting edges. *CoRR*, abs/1709.06805, 2017. arXiv:1709.06805.
- 27 Sarel Har-Peled, Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, Micha Sharir, and Max Willert. Stabbing pairwise intersecting disks by five points. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 50:1–50:12, 2018. doi:10.4230/LIPIcs.ISAAC.2018.50.
- 28 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.
- 29 Petr Hliněný and Jan Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discrete Mathematics*, 229(1-3):101–124, 2001. doi:10.1016/S0012-365X(00)00204-1.
- 30 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 31 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, December 2001.
- 32 Ross J. Kang and Tobias Müller. Sphere and Dot Product Representations of Graphs. *Discrete & Computational Geometry*, 47(3):548–568, 2012. doi:10.1007/s00454-012-9394-8.
- 33 Seog-Jin Kim, Alexandr Kostochka, and Kittikorn Nakprasit. On the chromatic number of intersection graphs of convex sets in the plane. *the electronic journal of combinatorics*, 11(1):52, 2004.
- 34 Paul Koebe. Kontaktprobleme der konformen Abbildung. *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physikalische Klasse*, 88:141–164, 1936.
- 35 Jan Kratochvíl. String graphs. II. recognizing string graphs is NP-hard. *J. Comb. Theory, Ser. B*, 52(1):67–78, 1991. doi:10.1016/0095-8956(91)90091-W.
- 36 Jan Kratochvíl and Jiří Matoušek. Intersection graphs of segments. *J. Comb. Theory, Ser. B*, 62(2):289–315, 1994. doi:10.1006/jctb.1994.1071.
- 37 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006. doi:10.1007/11847250\_14.
- 38 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *Algorithms - ESA 2015 - 23rd Annual European*

## 17:18 Maximum Clique in Disk-Like Intersection Graphs

- Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 865–877, 2015. doi:10.1007/978-3-662-48350-3\_72.
- 39 Terry A. McKee and Fred R. McMorris. *Topics in intersection graph theory*. SIAM, 1999.
  - 40 Matthias Middendorf and Frank Pfeiffer. The max clique problem in classes of string-graphs. *Discrete Mathematics*, 108(1-3):365–372, 1992. doi:10.1016/0012-365X(92)90688-C.
  - 41 Bernard M. E. Moret. Planar NAE3SAT is in P. *ACM SIGACT News*, 19(2):51–54, 1988.
  - 42 Dana Moshkovitz and Ran Raz. Sub-constant error probabilistically checkable proof of almost-linear size. *Computational Complexity*, 19(3):367–422, 2010. doi:10.1007/s00037-009-0278-0.
  - 43 Tim Nieberg and Johann Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. In *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, pages 296–306, 2005. doi:10.1007/11671411\_23.
  - 44 Tim Nieberg, Johann Hurink, and Walter Kern. A robust PTAS for maximum weight independent sets in unit disk graphs. In *Graph-Theoretic Concepts in Computer Science, 30th International Workshop, WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, pages 214–221, 2004. doi:10.1007/978-3-540-30559-0\_18.
  - 45 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.
  - 46 Vijay Raghavan and Jeremy P. Spinrad. Robust algorithms for restricted domains. *J. Algorithms*, 48(1):160–172, 2003. doi:10.1016/S0196-6774(03)00048-8.
  - 47 Marcus Schaefer, Eric Sedgwick, and Daniel Stefankovic. Recognizing string graphs in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003. doi:10.1016/S0022-0000(03)00045-X.
  - 48 Lajos Stachó. A solution of gallai’s problem on pinning down circles. *Mat. Lapok*, 32(1-3):19–47, 1981.
  - 49 Erik Jan van Leeuwen. Better approximation schemes for disk graphs. In *Algorithm Theory - SWAT 2006, 10th Scandinavian Workshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*, pages 316–327, 2006. doi:10.1007/11785293\_30.
  - 50 Erik Jan van Leeuwen. *Optimization and Approximation on Systems of Geometric Objects*. PhD thesis, Utrecht University, 2009.

# Parameterized Complexity of Feedback Vertex Sets on Hypergraphs

**Pratibha Choudhary**

Indian Institute of Technology Jodhpur, Jodhpur, India  
pratibhac247@gmail.com

**Lawqueen Kanesh**

Institute of Mathematical Sciences, HBNI, Chennai, India  
lawqueen@imsc.res.in

**Daniel Lokshtanov**

University of California Santa Barbara, Santa Barbara, USA  
daniello@ucsb.edu

**Fahad Panolan**

Indian Institute of Technology Hyderabad, India  
fahad@cse.iith.ac.in

**Saket Saurabh**

Institute of Mathematical Sciences, HBNI, Chennai, India  
University of Bergen, Norway  
saket@imsc.res.in

---

## Abstract

A *feedback vertex set* in a hypergraph  $H$  is a set of vertices  $S$  such that deleting  $S$  from  $H$  results in an acyclic hypergraph. Here, deleting a vertex means removing the vertex and all incident hyperedges, and a hypergraph is *acyclic* if its vertex-edge incidence graph is acyclic. We study the (parameterized complexity of) the HYPERGRAPH FEEDBACK VERTEX SET (HFVS) problem: given as input a hypergraph  $H$  and an integer  $k$ , determine whether  $H$  has a feedback vertex set of size at most  $k$ . It is easy to see that this problem generalizes the classic FEEDBACK VERTEX SET (FVS) problem on graphs. Remarkably, despite the central role of FVS in parameterized algorithms and complexity, the parameterized complexity of a generalization of FVS to hypergraphs has not been studied previously. In this paper, we fill this void. Our main results are as follows

- HFVS is  $W[2]$ -hard (as opposed to FVS, which is fixed parameter tractable).
- If the input hypergraph is restricted to a linear hypergraph (no two hyperedges intersect in more than one vertex), HFVS admits a randomized algorithm with running time  $2^{\mathcal{O}(k^3 \log k)} n^{\mathcal{O}(1)}$ .
- If the input hypergraph is restricted to a  $d$ -hypergraph (hyperedges have cardinality at most  $d$ ), then HFVS admits a deterministic algorithm with running time  $d^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ .

The algorithm for linear hypergraphs combines ideas from the randomized algorithm for FVS by Becker et al. [J. Artif. Intell. Res., 2000] with the branching algorithm for POINT LINE COVER by Langerman and Morin [Discrete & Computational Geometry, 2005].

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** feedback vertex sets, hypergraphs, FPT, randomized algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.18

**Funding** *Saket Saurabh*: Received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.



**Acknowledgements** We thank the anonymous referees of an earlier version of the paper. Their comments helped us a lot in improving the paper.



© Pratibha Choudhary, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh; licensed under Creative Commons License CC-BY  
40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 18; pp. 18:1–18:15



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

## 1 Introduction

It would be an understatement to say that VERTEX COVER (VC) and FEEDBACK VERTEX SET (FVS) have played a pivotal roles in the development of the field of Parameterized Complexity. VERTEX COVER asks if given an undirected graph  $G$  and a positive integer  $k$ , there exists a set  $S$  of  $k$  vertices which intersects every edge in  $G$ . FEEDBACK VERTEX SET asks if given an undirected graph  $G$  and a positive integer  $k$ , there exists a set  $S$  (called *feedback vertex set* or *in short fvs*) of  $k$  vertices which intersects every cycle in  $G$ . While there has been no improvement in the parameterized algorithm for VC in the last 14 years [9] (the conference version appeared in MFCS 2006), faster algorithms for FVS have been developed over the last decade. The best known algorithm for VC runs in time  $\mathcal{O}(1.2738^k + kn)$  [9]. On the other hand, for FVS, the first deterministic  $\mathcal{O}(c^k n^{\mathcal{O}(1)})$  algorithm was designed only in 2005; independently by Dehne et al. [13] and Guo et al. [20]. It is important to note here that a randomized algorithm for FVS with running time  $\mathcal{O}(4^k n^{\mathcal{O}(1)})$  [5] was known in as early as 1999. The deterministic algorithms led to the race of improving the base of the exponent for FVS algorithms and several algorithms [6, 7, 8, 11, 21, 25, 27], both deterministic and randomized, have been designed. Until few months ago the best known deterministic algorithm for FVS ran in time  $3.619^k n^{\mathcal{O}(1)}$  [25], while the Cut and Count technique by Cygan et al. [11] gave the best known randomized algorithm running in time  $3^k n^{\mathcal{O}(1)}$ . However, just in last few months both these algorithms have been improved; Iwata and Kobayashi [21, IPEC 2019] designed the fastest known deterministic algorithm with running time  $\mathcal{O}(3.460^k n)$  and Li and Nederlof [27, SODA 2020] designed the fastest known randomized algorithm with running time  $2.7^k n^{\mathcal{O}(1)}$ . We would like to remark that many variants of FVS have been studied in literature such as CONNECTED FVS [11, 31], INDEPENDENT FVS [2, 28, 30], SIMULTANEOUS FVS [4, 34] and SUBSET FVS [12, 22, 23, 24, 29].

The main objective of this paper is a study of FVS on hypergraphs. A hypergraph is a set family  $H$  with a universe  $V(H)$  and a family of hyperedges  $E(H)$ , where each hyperedge (or edge) is a subset of  $V(H)$ . If every hyperedge in  $E(H)$  is of size at most  $d$ , it is known as a  $d$ -hypergraph. Observe that if each hyperedge is of size *exactly* two, we get an undirected graph. The natural question is, how does VC generalize to hypergraphs. If  $(G, k)$  is an instance of VC, we can view VC as the following problem: Given a hypergraph with vertex set  $V(G)$  and the set of hyperedges  $E(G)$ , does there exist a set of  $k$  vertices that intersects every hyperedge. Thus, VC is a special case of HITTING SET (HS): Given a hypergraph  $H$  and a positive integer  $k$ , does there exist a set of  $k$  vertices that intersects every hyperedge. If the size of each hyperedge is upper bounded by  $d$ , we refer to the problem as the  $d$ -HITTING SET ( $d$ -HS) problem. Observe that VC is equivalent to the 2-HS problem. It is well known that HS does not admit an algorithm with running time  $f(k)n^{\mathcal{O}(1)}$ , where the function  $f$  depends only on  $k$  due to Exponential Time Hypothesis (ETH). That is, the problem is known to be W[2]-hard. On the other hand,  $d$ -HS is solvable in time  $d^k n^{\mathcal{O}(1)}$  and admits a kernel of size  $\mathcal{O}(k^d)$  [1, 17]. It is worth noting that  $d$ -HS does not admit a kernel of size  $\mathcal{O}(k^{d-\epsilon})$  under plausible complexity theory assumptions [14]. Thus, generalization of VC on hypergraphs is well studied. However, there is very little study of FVS on hypergraphs. The only known algorithmic result is a factor  $d$  approximation for FVS on  $d$ -hypergraphs [19]. Upper bounds on minimum fvs in 3-uniform linear hypergraphs are studied in [15].

The objective of this paper is to study the hypergraph variant of the FEEDBACK VERTEX SET problem from the viewpoint of Parameterized Complexity.

One of the main reasons for the lack of study of FVS on hypergraphs is that it is not as natural to define the generalization of FVS in hypergraphs, as it is for the case of VC (generalizing to HS and  $d$ -HS) in hypergraphs. To generalize the notion of fvs to hypergraphs, we need to have notions of *cycles* and *forests* in hypergraphs. For cycles, we use the same notion as that in graph theory [15]: a cycle in a hypergraph  $H$  is a sequence  $(v_0, e_0, v_1, \dots, v_\ell, e_\ell, v_0)$  such that  $v_0, \dots, v_\ell$  are distinct vertices,  $e_0, \dots, e_\ell$  are distinct hyperedges,  $\ell \geq 1$  and  $v_i, v_{(i+1) \bmod (\ell+1)} \in e_i$  for any  $i \in \{0, \dots, \ell\}$ . Given the above definition of cycle, a subset  $S$  of vertices in a hypergraph  $H$  is called a *feedback vertex set*, if there does not exist a cycle in the hypergraph obtained after *deleting* vertices in  $S$ . The next natural question is what do we mean by *deletion* of a vertex in a hypergraph. There are two ways to define the vertex deletion operation in hypergraphs:

1. *Strong deletion* or simply *deletion* of a vertex  $v$  implies deleting  $v$  along with all the hyperedges containing the vertex  $v$ .
2. *Weak deletion* of a vertex  $v$  implies deleting  $v$  without deleting the hyperedges that contain  $v$ . That is, the hypergraph  $H'$  obtained after weak deletion of a vertex  $v$  from  $H$  has vertex set  $V(H)$  and edge set  $\{e \in E(H) : v \notin e\} \cup \{e \setminus \{v\} : e \in E(H), v \in e, |e| > 2\}$ .

For a hypergraph  $H$  we use the notation  $H - S$  to denote the graph obtained after (weak/strong) deletion of the vertices in  $S$ . Consequently, there are two ways one may define the FEEDBACK VERTEX SET problem – WEAK FVS and STRONG FVS.

**Our Results and Methods.** Given a hypergraph  $H$ , the incidence graph  $G$  corresponding to  $H$  is the bipartite graph with bipartition  $V(G) = A \uplus B$  where  $A = V(H)$  and  $B = E(H)$ , and for any  $v \in V(H)$  and  $e \in E(H)$ ,  $ve$  is an edge in  $G$  if and only if  $v \in e$  in  $H$ . Observe that WEAK FVS corresponds to finding a fvs  $S$  in  $G$  of size at most  $k$ , such that  $S \subseteq A$  and  $G - S$  is a forest. Using the best known algorithm for WEIGHTED FVS [3] running in  $3.618^k n^{O(1)}$  time, we can solve WEAK FVS in  $3.618^k n^{O(1)}$  time, by transforming the problem to WEIGHTED FVS. To transform WEAK FVS to WEIGHTED FVS we assign every vertex in  $B$  a weight of  $k + 1$ , every vertex in  $A$  a weight of 1. Now the problem of finding an fvs of weight at most  $k$  will be equivalent to solving WEAK FVS for the original hypergraph. Thus WEAK FVS is not challenging as a parameterized problem.

Hence, we only consider FVS on hypergraphs with respect to *strong deletion*. In particular, we study HYPERGRAPH FEEDBACK VERTEX SET (HFVS). Here, given an  $n$ -vertex hypergraph  $H$  and a positive integer  $k$ , the objective is to check whether there exists a set  $S \subseteq V(H)$  of size at most  $k$ , such that  $H - S$  is acyclic. As in the case of HS, it is expected that HFVS is  $W[2]$ -hard and this can be proven using a parameter preserving reduction from SET COVER (which is “equivalent” to HS). We prove the following theorem in the full version of the paper.

► **Theorem 1** ( $\clubsuit^1$ ). *HFVS is  $W[2]$ -hard when parameterized by  $k$ .*

Theorem 1 is not surprising as a generalization of even VC to hypergraphs i.e. HS, is  $W[2]$ -hard.

<sup>1</sup> Proofs of results marked with  $\clubsuit$  can be found in the full version of the paper.

FVS is a deeply studied problem in Parameterized Complexity, and thus, we tried to generalize the existing algorithms as much as possible. However, considering the problem on general hypergraphs is pushing it too far (Theorem 1). This motivated us to look for families of hypergraphs, which are a strict generalization of graphs and where FVS turns out to be tractable. Specifically, we study the problem for the cases when the input is restricted to *linear hypergraphs* and *d-hypergraphs*.

A hypergraph  $H$  is linear if  $|e \cap e'| \leq 1$  for any two distinct hyperedges  $e, e' \in E(H)$ . We show that for both these families, HFVS admits fixed parameter tractable (FPT) algorithms. Our main result is a randomized algorithm for the case when the input hypergraph is linear, and the size of the hyperedges is not bounded. Thus our positive results are the following.

► **Theorem 2** (♣). *There exists a deterministic algorithm for HFVS on d-hypergraphs, running in time  $d^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ .*

► **Theorem 3**. *There exists an  $\mathcal{O}^*(2^{\mathcal{O}(k^3 \log k)})$  time<sup>2</sup> randomized algorithm for HFVS on linear hypergraphs, which produces a false negative output with probability at most  $\frac{1}{n^{\mathcal{O}(1)}}$ , and no false positive output.*

The restriction to linear hypergraphs corresponds to exclusion of  $C_4$  or  $K_{2,2}$  in the corresponding incidence graph.  $K_{i,j}$  refers to the complete bipartite graph with partitions of sizes  $i$  and  $j$ . There has been extensive work on RED-BLUE DOMINATING SET for  $K_{i,j}$  free graphs [10, 18, 32, 33]. Theorem 3 can be viewed as an analog of RED-BLUE DOMINATING SET results for  $K_{2,2}$  free graphs.

The starting point of both the above mentioned algorithms (Theorems 2 and 3) is recasting HFVS as an appropriate problem on the incidence graph  $G$  of the given hypergraph  $H$ . Proof of Theorem 3 starts with the observation that for any subset  $S \subseteq V(H)$ ,  $H - S$  is acyclic if and only if  $G - N_G[S]$  (notations defined in Section 2) is acyclic. Consequently, HFVS is same as the following problem (proof to be given in the full version of the paper).

DOMINATING FVS ON BIPARTITE GRAPHS (DFVSB)

**Parameter:**  $k$

**Input:** A bipartite graph  $G$  with bipartition  $V(G) = A \uplus B$  and  $k \in \mathbb{N}$ .

**Question:** Is there a subset  $S \subseteq A$  of size at most  $k$  such that  $G - N_G[S]$  is acyclic?

For a bipartite graph  $G = (A \uplus B, E)$ , we say that a subset  $S \subseteq A$  is a *dominating feedback vertex set* (dfvs) for  $G$  if  $G - N[S]$  is acyclic. Let  $G$  be the incidence graph of a hypergraph  $H$ . Then, notice that  $H$  is a  $d$ -hypergraph if and only if  $\max_{e \in E(H)} d_G(e) \leq d$ . Also,  $H$  is linear if and only if  $G$  is  $C_4$ -free. As a result HFVS on  $d$ -hypergraphs and linear hypergraphs are equivalent to DFVSB on bipartite graphs  $G = (A \uplus B, E)$  with  $\max_{w \in B} d(w) \leq d$  and on  $C_4$ -free bipartite graphs, respectively.

Theorem 2 shows that for  $d$ -hypergraphs, HFVS is similar to  $d$ -HS. Proof of Theorem 2 utilizes iterative compression. The compression step involves a branching strategy that uses a measure more generalized than the one used in known FVS algorithms for undirected graphs.

Our proof for Theorem 3 is inspired by the randomized algorithm of Becker et al. [5] that runs in  $\mathcal{O}(4^k n^{\mathcal{O}(1)})$  time and the branching algorithm for POINT LINE COVER by Langerman

<sup>2</sup> Polynomial dependency on  $n$  is hidden in  $\mathcal{O}^*$  notation.

and Morin [26]. The algorithm of Becker et al. [5] first preprocesses the input graph and transforms it into a graph with minimum degree at least 3 and then shows that for any fvs, at least half the edges in a preprocessed graph are incident to the vertex set of the fvs. This immediately gives the following algorithm: “pick an edge uniformly at random, then pick a vertex that is an endpoint of this edge uniformly at random and add it to a solution, and recurse”. Let  $G$  be the incidence graph of a hypergraph  $H$ . First we preprocess  $G$  and show that in the preprocessed graph (say  $G$ ) for any dfvs  $S$  of size at most  $k$ , at least  $1/\text{poly}(k)$  fraction of all the edges are incident to  $N[S]$ . Here,  $\text{poly}$  denotes a polynomial function. We call this property  $\alpha$ -covering, with  $\alpha$  being  $\text{poly}(k)$ . Let  $S$  be a fixed fvs of size at most  $k$ . We now compute the probability of finding  $S$ . Note that if we randomly pick an edge  $f$  (that is, pick an edge from graph  $G$  uniformly at random and then select  $f$  as the hyperedge incident to the selected edge), then with probability  $1/\text{poly}(k)$  there exists a vertex incident to  $f$  that is contained in  $S$ . However, unlike the case of FVS in graphs, here we cannot randomly select a vertex from  $f$ , as the size of  $f$  could be independent of  $k$ . However, for now let us assume that we can preprocess  $G - f$  such that the  $\alpha$ -covering property holds even after we delete  $f$  from  $G$ . We assume that  $\alpha$ -covering property holds recursively after each iteration of preprocessing. Suppose we do this process  $k^2 + 1$  times. Then we have a collection of hyperedges  $\mathcal{F} = \{f_1, \dots, f_{k^2+1}\}$  such that each of them has a non-trivial intersection with  $S$ . Observe that the pairwise intersection of these hyperedges cannot be more than one, since  $G$  excludes  $C_4$  as a subgraph ( $H$  being a linear hypergraph). However,  $S$  is a solution of size at most  $k$ , and hence there exist  $k + 1$  hyperedges  $f'_1, \dots, f'_{k+1}$  in  $\mathcal{F}$  such that  $|f'_i \cap f'_j| = \{v\}$ ,  $i \neq j$  for some  $v \in A = V(H)$ . This implies that  $v$  must belong to  $S$ , as each of  $f'_1, \dots, f'_{k+1}$  has a non-trivial intersection with  $S$  and if we don't pick  $v$ , then every solution is of size at least  $k + 1$ . Hence, we delete  $v$  along with all those edges in  $H$  that  $v$  participates in, and recursively find a solution of size  $k - 1$  in the reduced hypergraph.

However, unlike the case with FVS for graphs, in HFVS we cannot delete degree 1 vertices or contract degree 2 vertices directly. When we delete a hyperedge, we need to *remember* that we are seeking a solution that is a dfvs as well as a hitting set for the selected set. To implement this idea in our algorithm, we maintain a family  $\mathcal{F}$  such that our solution is a dfvs for  $G$  as well as a hitting set for  $\mathcal{F}$ . We exploit the fact that  $|\mathcal{F}| \leq k^2 + 1$  and design reduction rules to get rid of certain degree 1 vertices and shorten degree 2 paths, as well as caterpillars (defined later) like degree 2 paths. We can show that after these reduction rules are performed, the  $\alpha$ -covering property holds for the preprocessed graph,  $\alpha$  being  $\text{poly}(k)$ .

## 2 Preliminaries

For a positive integer  $\ell \in \mathbb{N}$ , we use  $[\ell]$  to denote the set  $\{1, 2, \dots, \ell\}$ . We use the term graph to denote a simple graph without multiple edges, loops and labels. For the notations related to graphs that are not explicitly stated here, we refer to the book [16]. For a graph  $G$  and a subset of vertices  $U \subseteq V(G)$ ,  $N_G(U)$  and  $N_G[U]$  denote the open neighborhood and closed neighborhood of  $U$ , respectively. That is,  $N_G(U) = \{v \in V(G) : u \in U \text{ and } uv \in E(G)\} \setminus U$  and  $N_G[U] = N_G(U) \cup U$ . If  $U = \{u\}$ , then we write  $N_G(u) = N_G(U)$  and  $N_G[u] = N_G[U]$ . Also, we omit the subscript  $G$ , if the graph in consideration is clear from the context. For a graph  $G$ , a vertex subset  $X \subseteq V(G)$ , and an edge subset  $F \subseteq E(G)$ , we use  $G[X]$ ,  $G - X$ , and  $G - F$  to denote the graph induced by  $X$ , the graph induced by  $V(G) \setminus X$ , and the graph with vertex set  $V(G)$  and edge set  $E(G) \setminus F$ , respectively. Moreover, if  $X = \{v\}$ , then we write  $G - v = G - X$ . For a graph  $G$ ,  $X, Y \subseteq V(G)$ , and  $X \cap Y = \emptyset$ ,  $E(X, Y) \subseteq E(G)$  denotes the set of edges in  $G$  whose one endpoint is in  $X$  and the other one is in  $Y$ . For a

graph  $G$  and a non-edge  $uv$  in  $G$ , we use  $G + uv$  to denote the graph with vertex set  $V(G)$  and edge set  $E(G) \cup \{uv\}$ . A path  $P$  in a graph  $G$  is a sequence of distinct vertices  $u_1 \dots u_\ell$  such that for all  $i \in [\ell - 1]$ ,  $u_i u_{i+1} \in E(G)$ . We say that a path  $P = u_1 \dots u_\ell$  in a graph  $G$  is a *degree two path* in  $G$ , if for each  $i \in [\ell]$ , the degree of  $u_i$  in  $G$ , denoted by  $d_G(u_i)$ , is equal to 2. For a path/cycle  $P$ , we use  $V(P)$  to denote the set of vertices present in  $P$ . A triangle is a cycle consisting of exactly 3 edges. A bipartite graph  $G = (A \uplus B, E)$  is called a  $d$ -bipartite graph if  $d_G(b) \leq d$  for all  $b \in B$ . For two hypergraphs  $H_1$  and  $H_2$ ,  $H_1 \cup H_2$  denotes the hypergraph with the vertex set  $V(H_1) \cup V(H_2)$  and the edge set  $E(H_1) \cup E(H_2)$ .

### 3 Feedback Vertex Sets on Linear Hypergraphs

In this section we design an FPT algorithm for HFVS on linear hypergraphs. Towards this, we prove the following result about DFVSB, from which Theorem 3 follows as a corollary.

► **Theorem 4.** *There exists an  $\mathcal{O}^*(2^{\mathcal{O}(k^3 \log k)})$  time randomized algorithm for DFVSB on  $C_4$ -free bipartite graphs, which produces a false negative output with probability at most  $\frac{1}{n^{\mathcal{O}(1)}}$ , and no false positive output.*

To prove Theorem 4, we first define a few generalizations of these problems that appear naturally in the recursive steps. Let  $\mathcal{F}$  be a family of sets over a universe  $A$ , then we define a bipartite graph  $G_{\mathcal{F}}$  as follows. Let the bipartition of  $V(G_{\mathcal{F}})$  be  $A_{\mathcal{F}} \uplus B_{\mathcal{F}}$ , where  $A_{\mathcal{F}} = A$  and  $B_{\mathcal{F}} = \mathcal{F}$ . Edge set  $E(G_{\mathcal{F}}) = \{\{u, Y\} : u \in A, u \in Y \in \mathcal{F}\}$ . Let  $G$  be a  $C_4$  free bipartite graph with bipartition  $V(G) = A \uplus B$ , and  $\mathcal{F}$  be a family of sets over the universe  $A$ . We define the graph  $G \cup G_{\mathcal{F}} = (A^* \uplus B^*, E^*)$  as follows. Let  $A^* = A, B^* = B \uplus B_{\mathcal{F}}$  and  $E^* = E(G) \cup E(G_{\mathcal{F}})$ . The following problem generalizes HFVS on linear hypergraphs.

HITTING HYPERGRAPH FEEDBACK VERTEX SET (HHFVS)      **Parameter:**  $k + |E(H_2)|$   
**Input:** Two linear hypergraphs  $H_1, H_2$  such that  $V(H_1) = V(H_2)$ ,  $E(H_1) \cap E(H_2) = \emptyset$ , and  $H_1 \cup H_2$  is a linear hypergraph,  $k \in \mathbb{N}$ .  
**Question:** Does there exist a set  $S \subseteq V(H_1)$  of size at most  $k$ , such that  $H_1 - S$  is acyclic and  $S$  is a hitting set for  $E(H_2)$ ?

Observe that, if  $H_2 = \emptyset$ , HHFVS is the same as HFVS (for linear hypergraphs). Next, we define the “graph” version of HHFVS, which generalizes DFVSB on  $C_4$ -free graphs.

HITTING DOMINATING BIPARTITE FVS (HDBFVS)      **Parameter:**  $k + |\mathcal{F}|$   
**Input:** A  $C_4$  free bipartite graph  $G$  with bipartition  $V(G) = A \uplus B$ , a family  $\mathcal{F}$  of subsets of  $A$  such that the graph  $G \cup G_{\mathcal{F}}$  is a  $C_4$  free bipartite graph,  $k \in \mathbb{N}$ .  
**Question:** Does there exist a set  $S \subseteq A$  of size at most  $k$ , such that  $G - N[S]$  is a forest and  $S$  is a hitting set for  $\mathcal{F}$ ?

We say that an instance  $(G = (A \uplus B, E), \mathcal{F}, k)$  is a *valid instance* of HDBFVS, if  $\mathcal{F}$  is a family of subsets of  $A$  such that the graph  $G \cup G_{\mathcal{F}}$  is a  $C_4$ -free bipartite graph.

In the rest of the section, whenever we say  $\mathcal{I} = (G = (A \uplus B, E), \mathcal{F}, k)$  is an instance of HDBFVS, it implies that  $\mathcal{I}$  is a valid instance of HDBFVS. Further, after each application of a reduction rule, we ensure that the instance remains valid.

The proof of the following simple observation follows from the fact that  $G \cup G_{\mathcal{F}}$  is  $C_4$ -free.

► **Observation 3.1.** *If  $(G = (A \uplus B, E), \mathcal{F}, k)$  is an instance of HDBFVS, then (i) pairwise intersection of sets in  $\mathcal{F}$  is of size at most 1, and (ii) for every vertex  $b \in B$  and  $F \in \mathcal{F}$ ,  $|N(b) \cap F|$  is at most one.*

Given an instance  $(H_1, H_2, k)$  of HHFVS, we can obtain an instance,  $(G, \mathcal{F}, k)$ , of HDBFVS in a canonical way. Next lemma shows their equivalence.

► **Lemma 5 (♣).**  *$(H_1, H_2, k)$  is a YES-instance of HHFVS if and only if  $(G, \mathcal{F} = E(H_2), k)$  is a YES-instance of HDBFVS, where  $G$  is the incidence graph of the hypergraph  $H_1$ .*

The rest of the section is devoted to designing an FPT algorithm for HDBFVS. Given an instance  $(G = (A \uplus B, E), \mathcal{F}, k)$  of HDBFVS, we first define some notations. For a vertex  $v \in A$ ,  $X_v$  denotes the set  $\{Y \mid Y \in \mathcal{F}, v \in Y\}$ . We *distinguish* the vertices in  $A$  as follows.

- If  $|X_v| \geq 2$ , i.e.,  $v$  is in at least two sets in  $\mathcal{F}$ , then we say that  $v$  is a *special* vertex.
- If  $|X_v| = 1$ , i.e.,  $v$  is in exactly one set in  $\mathcal{F}$ , then we say that  $v$  is an *easy* vertex.
- Otherwise, we say that  $v$  is a *trivial* vertex.

Let  $V(\mathcal{F}) = \{v \in A \mid v \in Y \text{ where } Y \in \mathcal{F}\}$ . For a graph  $G^*$ , the notations  $V_0(G^*)$ ,  $V_{=1}(G^*)$ ,  $V_{=2}(G^*)$ , and  $V_{\geq 3}(G^*)$  denote the set of isolated vertices, the set of vertices of degree 1, the set of vertices of degree 2, and the set of vertices of degree at least 3 in  $G^*$ , respectively.

► **Lemma 6.** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS. Then, the number of special vertices in  $A$  is upper bounded by  $\binom{|\mathcal{F}|}{2}$ .*

**Proof.** For contradiction, assume that the number of special vertices in  $A$  is more than  $\binom{|\mathcal{F}|}{2}$ . By pigeonhole principle there exist two special vertices  $u, v \in A$ , such that  $|X_u \cap X_v| \geq 2$ . Let  $Y_1, Y_2 \in X_u \cap X_v$ . This implies that  $u, v \in Y_1 \cap Y_2$ , contradicting Observation 3.1(i). ◀

Now we state some reduction rules that are applied exhaustively by the algorithm in the order in which they appear. Let  $(G, \mathcal{F}, k)$  be an instance of HDBFVS and  $(G', \mathcal{F}', k)$  be the resultant instance after application of a reduction rule. To show that a reduction rule is safe, we will prove that  $(G, \mathcal{F}, k)$  is a YES-instance if and only if  $(G', \mathcal{F}', k)$  is a YES-instance.

► **Reduction Rule 3.1.** *If one of the following holds, then return a trivial NO-instance: (i)  $k < 0$ ; (ii)  $k = 0$  and  $G$  is not acyclic; and (iii)  $k = 0$  and  $\mathcal{F}$  is not empty.*

► **Reduction Rule 3.2.** *If  $k \geq 0$ ,  $G$  is acyclic and  $\mathcal{F}$  is empty, then return a trivial YES-instance.*

► **Reduction Rule 3.3.** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $b \in B$  be a vertex that does not participate in any cycle in  $G$ . Then, output  $(G - b, \mathcal{F}, k)$ .*

► **Reduction Rule 3.4.** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $v \in A$  be an isolated vertex in  $G$ . If  $v$  is a trivial vertex, then output  $(G - v, \mathcal{F}, k)$ .*

It is easy to see that the above reduction rules are safe and can be applied in polynomial time. Observe that, when Reduction Rules 3.3 and 3.4 are no longer applicable, then  $V_0(G) \subseteq A$  and each isolated vertex in  $G$  is either easy or special. Next, we state a reduction rule that will help to bound the number of easy isolated vertices in  $G$ .

► **Reduction Rule 3.5 ( $\star^3$ ).** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $v \in A$  be an isolated vertex in  $G$ . Suppose  $v$  is an easy vertex,  $X_v = \{Y\}$ , and  $|Y| > 1$ . Then output  $(G', \mathcal{F}', k)$ , where  $G' = G - v$  and  $\mathcal{F}' = (\mathcal{F} \setminus \{Y\}) \cup \{(Y \setminus \{v\})\}$ .*

<sup>3</sup> The safeness proofs of reduction rules marked with  $\star$  can be found in the full version of the paper.

► **Reduction Rule 3.6** ( $\star$ ). Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $v \in A$  be a vertex of degree 1 in  $G$ . If  $v$  is a trivial vertex, then output  $(G' = G - v, \mathcal{F}, k)$ .

Observe that when Reduction Rules 3.1 to 3.6 are no longer applicable, the following holds.

► **Lemma 7**. Let  $(G, \mathcal{F}, k)$  be an instance reduced with respect to Reduction Rules 3.1 to 3.6. Then, the following holds.

1.  $V_0(G) \cup V_{=1}(G) \subseteq A$ , all vertices in  $V_0(G) \cup V_{=1}(G)$  are either easy or special.
2.  $|V_0(G)| \leq |\mathcal{F}| + \binom{|\mathcal{F}|}{2}$ .

► **Lemma 8**. For any vertex  $b \in B$ ,  $|N_G(b) \cap V_{=1}(G)| \leq |\mathcal{F}|$ .

**Proof.** If there exists a vertex  $v \in N_G(b) \cap V_{=1}(G)$  which is a trivial vertex, then Reduction Rule 3.6 is applicable. Thus, (i) for all  $v \in N_G(b) \cap V_{=1}(G)$ ,  $v$  belongs to some set in  $\mathcal{F}$ . For contradiction, let  $b \in B$  be a vertex such that  $N_G(b)$  contains at least  $|\mathcal{F}| + 1$  vertices of degree 1 in  $G$ . Then, by pigeonhole principle and statement (i), at least two degree 1 vertices say  $u, v \in N_G(b)$  are contained in a set  $Y \in \mathcal{F}$ , which is a contradiction to item (ii) of Observation 3.1. This completes the proof of the lemma. ◀

Recall that,  $P$  is a degree two path in  $G$  if each vertex in  $P$  has degree exactly two in  $G$ . Next we state the reduction rules that help us bound the length of long degree two paths in  $G - V_{=1}(G)$ , i.e., to bound the length of degree two paths in the graph obtained after deleting vertices of degree 1 from  $G$ . Towards this, we first define the notion of a *nice path*.

► **Definition 9**. We say that  $P$  is a nice path in  $G$ , if  $P$  does not have any special vertex and the degree of each vertex in  $P$  in the graph  $G - V_{=1}(G)$  is exactly 2. A nice path  $P$  in  $G$  is a degree two nice path if each vertex in  $P$  has degree exactly 2 in  $G$ .

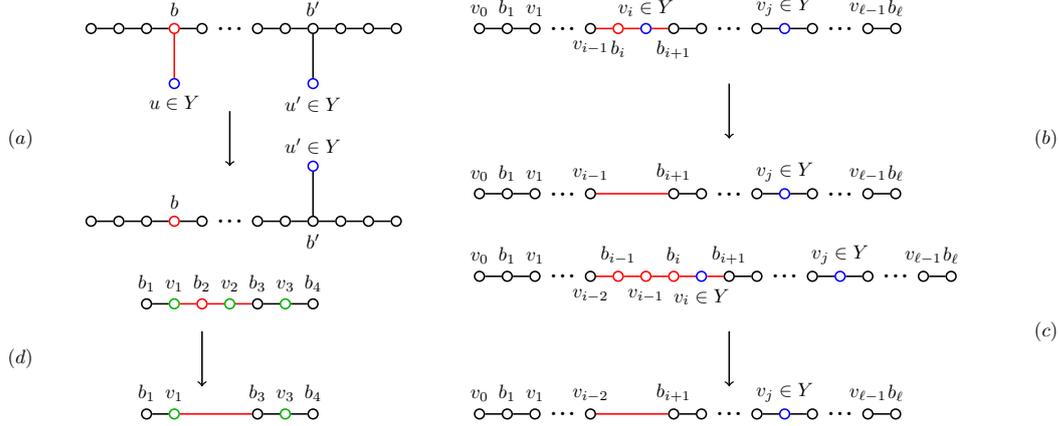
► **Reduction Rule 3.7** ( $\star$ ). Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS,  $P$  be a nice path in  $G$  and  $b, b' \in B$  be two vertices in  $P$ . If there exist two easy vertices  $u, u'$  whose degree is 1 in  $G$ , adjacent to  $b, b'$ , respectively, such that  $X_u = X_{u'} = \{Y\}$ , then return  $(G', \mathcal{F}', k)$ , where  $G' = G - u$ ,  $\mathcal{F}' = (\mathcal{F} \setminus \{Y\}) \cup \{Y \setminus \{u\}\}$ .

► **Lemma 10**. Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS reduced with respect to Reduction Rules 3.1 to 3.7. Then, in any nice path  $P$  in  $G$ , the number of vertices that are adjacent to a vertex of degree 1 in  $G$  is bounded by  $\binom{|\mathcal{F}|}{2} + |\mathcal{F}|$ .

**Proof.** From statement 1 in Lemma 7, we have that  $V_{=1}(G) \subseteq A$ . This implies,  $N_G(V_{=1}(G)) \subseteq B$ . Also, each vertex in  $V_{=1}(G)$  is either easy or special. By Lemma 6, the number of vertices that are special is bounded by  $\binom{|\mathcal{F}|}{2}$ . Therefore, the number of vertices in  $P$  that are adjacent to special degree 1 vertices is at most  $\binom{|\mathcal{F}|}{2}$ . Since Reduction Rule 3.7 is no longer applicable, we have that corresponding to each set  $Y \in \mathcal{F}$ , there exists at most 1 vertex in  $P$  that has a degree 1 neighbor  $u$  such that  $X_u = \{Y\}$ . This implies that at most  $|\mathcal{F}|$  vertices in  $P$  can be adjacent to degree 1 easy vertices, resulting in the mentioned upper bound. ◀

The next reduction rule helps us in upper bounding the length of degree two paths in  $G$ .

► **Reduction Rule 3.8** ( $\star$ ). Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $P = v_0 b_1 v_1 \dots v_{\ell-1} b_\ell$  be a degree two nice path in  $G$ , where  $\{b_1, \dots, b_\ell\} \subseteq B$ ,  $\{v_0, \dots, v_{\ell-1}\} \subseteq A$ , and  $\ell \geq 5$ . Let  $v_i, v_j \in A \cap (V(P) \setminus \{v_0, v_1\})$  be two distinct easy vertices such that  $X_{v_i} = X_{v_j} = \{Y\}$  for some  $Y \in \mathcal{F}$  and  $i < j$ . Then, return  $(G', \mathcal{F}', k)$ , where  $G'$  and  $\mathcal{F}'$  are defined as follows.



■ **Figure 1** (a) is an illustration of Reduction Rule 3.7, (b) and (c) are illustrations of two cases of Reduction Rule 3.8, (d) is an illustration of Reduction Rule 3.9. In (a), (b) and (c) *blue* vertices denote *easy* vertices, and in (d) *green* vertices denote *trivial* vertices.

- If  $X_{v_{i-1}} \neq X_{v_{i+1}}$  or  $X_{v_{i-1}} = X_{v_{i+1}} = \emptyset$ , then let  $G' = (G - \{b_i, v_i\}) + v_{i-1}b_{i+1}$  (i.e.,  $G'$  be the graph obtained by deleting the vertices  $b_i, v_i$  from  $G$  and by adding a new edge  $v_{i-1}b_{i+1}$ ) and  $\mathcal{F}' = (\mathcal{F} \setminus \{Y\}) \cup \{Y \setminus \{v_i\}\}$ .
- Otherwise,  $X_{v_{i-1}} = X_{v_{i+1}} = \{Y^*\}$ , then let  $G' = (G - \{b_{i-1}, v_{i-1}, b_i, v_i\}) + v_{i-2}b_{i+1}$  (i.e.,  $G'$  be the graph obtained by deleting the vertices  $b_{i-1}, v_{i-1}, b_i, v_i$  from  $G$  and by adding a new edge  $v_{i-2}b_{i+1}$ ) and  $\mathcal{F}' = (\mathcal{F} \setminus \{Y, Y^*\}) \cup \{Y^* \setminus \{v_{i-1}\}, Y \setminus \{v_i\}\}$ .

Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS reduced with respect to Reduction Rules 3.1 to 3.8. Observe that, for each set  $Y \in \mathcal{F}$  and a degree two nice path  $P$  in  $G$ , the number of easy vertices among the last  $|V(P)| - 3$  vertices in  $V(P)$  that belong to  $Y$ , is upper bounded by one. Reduction Rule 3.8 leads us to the following observation.

► **Observation 3.2.** Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be a reduced instance of HDBFVS with respect to Reduction Rules 3.1 to 3.8. Then, in any degree two nice path  $P$  of length at least 10 in  $G$ , the number of easy vertices is bounded by  $|\mathcal{F}| + 2$ .

► **Reduction Rule 3.9** ( $\star$ ). Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and  $P = b_1v_1b_2v_2b_3v_3b_4$  be a degree two nice path in  $G$ , such that  $\{b_1, \dots, b_4\} \subseteq B$ ,  $\{v_1, v_2, v_3\} \subseteq A$  and  $v_1, v_2, v_3$  are trivial vertices. Then, return  $(G', \mathcal{F}, k)$ , where  $G'$  is the graph obtained by deleting the vertices  $b_2, v_2$  from  $G$  and adding a new edge  $v_1b_3$  (i.e.,  $G' = (G - \{v_2, b_2\}) + v_1b_3$ ).

► **Observation 3.3.** Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS and let  $(G' = (A' \uplus B', E'), \mathcal{F}', k')$  be the reduced instance of HDBFVS obtained from  $(G = (A \uplus B, E), \mathcal{F}, k)$ , by exhaustive applications of Reduction Rules 3.1 to 3.9. Then,  $|\mathcal{F}'| = |\mathcal{F}|$  and  $k' \leq k$ .

We now bound the size of degree 2 path, when there is no degree 1 vertex in the graph.

► **Lemma 11** ( $\clubsuit$ ). Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS reduced with respect to Reduction Rules 3.1 to 3.9. Then, the number of vertices in a degree two path  $P$  in  $G - V_{=1}(G)$  is bounded by  $63|\mathcal{F}|^5 + 21$ .

From now on, we say that  $(G = (A \uplus B, E), \mathcal{F}, k)$  is a *reduced instance* of HDBFVS if it is reduced with respect to Reduction Rules 3.1 to 3.9. In the following lemma, we observe that, if  $(G = (A \uplus B, E), \mathcal{F}, k)$  is a YES-instance of HDBFVS, then a large number of edges in  $G$  is incident to the neighborhood of the solution.

► **Lemma 12.** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be a reduced instance of HDBFVS where  $G$  is not a forest. Then, for any solution  $S$ , at least  $1/(445|\mathcal{F}|^6 + 68)$  fraction of the total edges in  $E$  are incident to  $N[S]$ .*

**Proof.** Let  $E_S$  be the set of edges incident to all the vertices of  $N[S]$  in  $G$ . Observe that,  $E(G) = E_S \uplus E(G - N[S])$ . Since  $G - N[S]$  is a forest, we have that  $|E(G - N[S])| < |V(G - (N[S] \cup V_0(G)))|$ . We aim to show that  $|V(G - (N[S] \cup V_0(G)))| \leq (445|\mathcal{F}|^6 + 67) \cdot |E_S|$ . Let  $V^*$  be the set of vertices of degree 1 in  $G - N[S]$ . Let  $V_1^* \subseteq V^*$  be the set of vertices that have some neighbor in  $N[S]$  and  $V_2^* = V^* \setminus V_1^*$ . That is,  $V_2^* \subseteq V_{=1}(G)$ . Since the vertices in  $V_1^*$  have neighbors in  $N[S]$ , they contribute at least one edge to the set  $E_S$  and these edges are distinct. Hence,  $|V_1^*| \leq |E_S|$ .

Since  $V_2^* \subseteq V_{=1}(G)$ , by Lemma 7, we have that  $V_2^* \subseteq A$ . Thus,  $V_2^*$  have neighbors only in the set  $B \cap V(G - N[S])$ . Also, by Lemma 8, any vertex in  $B$  can be adjacent to at most  $|\mathcal{F}|$  vertices of degree 1 in  $G$ . Hence, each vertex in  $B \cap V(G - N[S])$  can be adjacent to at most  $|\mathcal{F}|$  vertices of  $V_2^*$ . Thus, we have that  $|V_2^*| \leq |\mathcal{F}| \cdot |B \cap V(G - N[S])|$ . Let  $G'$  be the graph  $G - (V_0(G) \cup V_2^*)$ . Since  $V_0(G) \cup V_2^* \subseteq A$ , we have that,  $B \subseteq V(G')$  and  $B \cap V(G - N[S]) = B \cap V(G' - N[S])$ . Hence, we obtain the following.

$$|V_2^*| \leq |\mathcal{F}| \cdot |B \cap V(G' - N[S])| \leq |\mathcal{F}| \cdot |V(G' - N[S])| \quad (1)$$

$$|V^*| = |V_1^*| + |V_2^*| \leq |\mathcal{F}| \cdot |V(G' - N[S])| + |E_S| \quad (\text{By (1) and } |V_1^*| \leq |E_S|) \quad (2)$$

Since the graph  $G'$  is obtained from  $G$  by deleting a subset of vertices that are contained in  $V_0(G) \cup V_{=1}(G) \subseteq A$ , the vertices that are degree 1 in  $G' - N[S]$  are either degree 1 vertices in  $G - N[S]$  and are contained in  $A$ , in particular in  $V_1^*$ , or they are contained in  $B$  and are neighbors of vertices in  $V_2^*$  in  $G$ . Let  $L$  be the set of leaves (vertices of degree 1) in  $G' - N[S]$ . We claim that  $L = V_1^*$ . For contradiction, assume that a vertex  $b \in B \cap L$ . Since Reduction Rule 3.3 is no longer applicable, we have that each vertex in  $B$  participates in a cycle in  $G$  and hence, participates in a cycle in  $G'$ . Therefore, degree of  $b$  is at least 2 in  $G'$ . Observe that  $b$  cannot have a neighbor in  $S$ , otherwise  $b \in N[S]$ . This implies that  $b$  has 2 neighbors in  $G' - N[S]$ , which contradicts that  $b \in L$ . Observe that each vertex in  $V_1^*$  is a leaf vertex in  $G' - N[S]$ . Hence  $L = V_1^*$ . Therefore, we obtain the following.

$$|L| \leq |E_S|. \quad (3)$$

$$V_{\geq 3}(G' - N[S]) \leq |E_S| \quad (\text{Since, } G' - N[S] \text{ is a forest, } V_{\geq 3}(G' - N[S]) \leq |L|) \quad (4)$$

Next we bound  $|V_0(G' - N[S])|$ . Since, for any vertex  $v$  in  $G' - N[S]$ ,  $d_G(v) \geq 1$ , we have that any vertex  $w \in V_0(G' - N[S])$  is adjacent to some vertex in  $N[S]$ . Then, each vertex in  $V_0(G' - N[S])$  contributes at least 1 edge to the set  $E_S$  and these edges are distinct.

$$\text{Therefore, } |V_0(G' - N[S])| \leq |E_S|. \quad (5)$$

Let  $V_{=2}^1(G')$  be the set of vertices of degree 2 in  $G' - N[S]$  that have a neighbor in  $N[S]$ . Then, each vertex in  $V_{=2}^1(G')$  contributes at least 1 edge to the set  $E_S$ . Therefore, we have

$$|V_{=2}^1(G')| \leq |E_S|. \quad (6)$$

Let  $V_{\neq 2}^2(G')$  be the set of vertices of degree 2 in  $G' - N[S]$ , that do not have a neighbor in  $N[S]$ . Then, each vertex in  $V_{\neq 2}^2(G')$  is contained in some maximal degree two path not containing any vertex of  $V_{=2}^1(G')$  in  $G' - N[S]$ . Observe that, since  $G' - N[S]$  is a forest, (i) the number of maximal degree two paths not containing any vertex of  $V_{=2}^1(G')$  in  $G' - N[S]$  is bounded by  $|L \cup V_{\geq 3}(G') \cup V_{\neq 2}^1(G')|$  and hence bounded by  $3|E_S|$  (because of (3),(4), and (6)). Observe that a degree two path not containing any vertex of  $V_{=2}^1(G')$  in  $G' - N[S]$  is

also a degree two path in  $G - V_{=1}(G)$ . By Lemma 11, (ii) the number of vertices in a degree two path in  $G - V_{=1}(G)$  is bounded by  $63|\mathcal{F}|^5 + 21$ . So, statements (i) and (ii) imply that

$$|V_{=2}^2(G')| \leq (189|\mathcal{F}|^5 + 63)|E_S| \quad (7)$$

Observe that  $V_{=2}(G' - N[S]) = V_{=2}^1(G') \cup V_{=2}^2(G')$ . By (6) and (7), we get the following.

$$|V_{=2}(G' - N[S])| = |V_{=2}^1(G')| + |V_{=2}^2(G')| \leq (189|\mathcal{F}|^5 + 64)|E_S| \quad (8)$$

Note that,  $V(G' - N[S]) = V_0(G' - N[S]) \cup L \cup V_{\geq 3}(G' - N[S]) \cup V_{=2}(G' - N[S])$ . Hence, we obtain the following using (3), (5), (4), and (8).

$$\begin{aligned} |V(G' - N[S])| &= |V_0(G' - N[S])| + |L| + |V_{\geq 3}(G' - N[S])| + |V_{=2}(G' - N[S])| \\ &\leq |E_S| + |E_S| + |E_S| + (189|\mathcal{F}|^5 + 64)|E_S| \\ &\leq (189|\mathcal{F}|^5 + 67)|E_S| \end{aligned} \quad (9)$$

Using (1) and (9), we obtain the following.

$$\begin{aligned} |V(G - (N[S] \cup V_0(G)))| &\leq |V(G' - N[S])| + |V_2^*| \\ &\leq (|\mathcal{F}| + 1)|V(G' - N[S])| \quad (\text{By (1)}) \\ &\leq (|\mathcal{F}| + 1)((189|\mathcal{F}|^5 + 67)|E_S|) \\ &\leq (445|\mathcal{F}|^6 + 67)|E_S| \end{aligned}$$

$$\begin{aligned} \text{Thus, } |E(G)| &= |E_S| + |E(G - N[S])| \\ &\leq |E_S| + |V(G - (N[S] \cup V_0(G)))| \leq (445|\mathcal{F}|^6 + 68)|E_S|. \end{aligned}$$

This concludes the proof.  $\blacktriangleleft$

► **Lemma 13.** *Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an instance of HDBFVS, where  $G$  is a forest and  $|\mathcal{F}| \leq k^2$ . Then, there exists an algorithm which solves the instance in  $\mathcal{O}^*((2k^4)^k)$  time.*

**Proof.** The Algorithm first applies Reduction Rules 3.1 to 3.9 exhaustively in the order in which they are stated. If any reduction rule solves the instance, then output YES and NO accordingly. All the reduction rules are safe, and can be applied in polynomial time, and they can be applied only polynomial many times since each reduction rule decreases the size of the graph. Let  $(G' = (A' \uplus B', E'), \mathcal{F}', k')$  be the reduced instance. Since Reduction Rule 3.3 is no longer applicable,  $B' = \emptyset$ , and hence  $G'$  is an edgeless graph with vertex set  $A'$ . By Lemma 7,  $|V(G')| = |A'| \leq |\mathcal{F}'| + \binom{|\mathcal{F}'|}{2}$ . By Observation 3.3, we have that  $|\mathcal{F}'| = |\mathcal{F}| \leq k^2$  and hence,  $|V(G')| \leq 2k^4$ . We enumerate all the subsets of  $V(G')$  of size at most  $k$  and check if they form a solution; else return a NO-instance. The algorithm runs in time  $\binom{2k^4}{k} n^{\mathcal{O}(1)} = \mathcal{O}^*((2k^4)^k)$ . This completes the proof.  $\blacktriangleleft$

► **Lemma 14.** *There is a randomized algorithm that takes an instance  $(G = (A \uplus B, E), \mathcal{F}, k)$  of HDBFVS as input, runs in  $\mathcal{O}^*((2k^4)^k)$  time, and outputs either YES, or NO, or an instance  $(G^* = (A^* \uplus B^*, E^*), \mathcal{F}^*, k^*)$  of HDBFVS where  $k^* < k$ , with the following guarantee.*

- *If  $(G, \mathcal{F}, k)$  is a YES-instance, then the output is YES or an equivalent YES-instance  $(G^*, \mathcal{F}^*, k^*)$  where  $k^* < k$ , with probability at least  $(445k^{12} + 68)^{-(k^2+1)}$ .*
- *If  $(G, \mathcal{F}, k)$  is a NO-instance, then the output is NO or an equivalent NO-instance  $(G^*, \mathcal{F}^*, k^*)$  where  $k^* < k$ , with probability 1.*

**Proof.** Let  $(G = (A \uplus B, E), \mathcal{F}, k)$  be an input instance of HDBFVS. Recall that, for any  $v \in A$ ,  $X_v = \{F \in \mathcal{F} : v \in F\}$ . The algorithm applies the following iterative procedure.

## 18:12 FVS in Hypergraphs

- Step 1.** If  $G$  is acyclic and  $|\mathcal{F}| \leq k^2$ , then apply Lemma 13 and solve the instance.
- Step 2.** If  $|\mathcal{F}| \geq k^2 + 1$ ;
- (i) If there exists a vertex  $v$  such that  $|X_v| \geq k + 1$ , return  $(G - N[v], \mathcal{F} \setminus X_v, k - 1)$ .
  - (ii) Otherwise, return that  $(G = (A \uplus B, E), \mathcal{F}, k)$  is a NO-instance of HDBFVS.
- Step 3.** Apply Reduction Rules 3.1 to 3.9 exhaustively in the order in which they are stated. If any reduction rule solves the instance, then output YES and NO accordingly. Let  $(G' = (A' \uplus B', E'), \mathcal{F}', k')$  be the reduced instance.
- Step 4.** Pick an edge  $e = ub$  in  $E(G')$  uniformly at random, where  $u \in A', b \in B'$ . Set  $G := G' - b, \mathcal{F} := \mathcal{F}' \cup \{N_{G'}(b)\}$ , and  $k := k'$ . Go to Step 1.

Now we prove the correctness of the algorithm. Correctness of Step 1 follows from Lemma 13. Next assume that  $|\mathcal{F}| \geq k^2 + 1$ . Let  $v$  be a vertex that is contained in at least  $k + 1$  sets in  $\mathcal{F}$ . By Observation 3.1, pairwise intersection of two sets in  $\mathcal{F}$  is at most 1. Thus, if we do not pick  $v$  in our solution, then we have to pick at least  $k + 1$  vertices to hit the sets in  $X_v$ . Thus  $v$  belongs to every solution of  $(G, \mathcal{F}, k)$  of HDBFVS. Hence,  $(G, \mathcal{F}, k)$  is a YES-instance of HDBFVS if and only if  $(G - v, \mathcal{F} \setminus X_v, k - 1)$  is a YES-instance of HDBFVS, and correctness of Step 2i follows. Suppose each vertex in  $A$  is contained in at most  $k$  sets of  $\mathcal{F}$ . Thus no set of size at most  $k$  can hit  $k^2 + 1$  sets of  $\mathcal{F}$ . Hence,  $(G, \mathcal{F}, k)$  is a NO-instance of HDBFVS, and correctness of Step 2ii follows. Correctness of the Step 3 is implied by the safeness of reduction rules. Suppose the algorithm does not stop in Step 3. Let  $(G', \mathcal{F}', k')$  be the reduced instance, where  $k' \leq k$ . Now, let  $S$  be a hypothetical solution to  $(G', \mathcal{F}', k')$ . By Lemma 12, the picked edge  $e = ub$  is incident to a vertex in  $N_{G'}[S]$  with probability at least  $1/(445|\mathcal{F}|^6 + 68)$ . This implies that with probability at least  $1/(445|\mathcal{F}|^6 + 68)$  a vertex in  $N_{G'}(b)$  is contained in  $S$ . Hence, if  $(G', \mathcal{F}', k')$  is a YES-instance, then  $(G' - b, \mathcal{F}' \cup \{N_{G'}(b)\}, k')$  is a YES-instance, with probability at least  $1/(445|\mathcal{F}|^6 + 68)$ . Also, notice that any solution to  $(G' - b, \mathcal{F}' \cup \{N_{G'}(b)\}, k')$  is also a solution to  $(G', \mathcal{F}', k')$ . Hence, if  $(G', \mathcal{F}', k')$  is a NO-instance, then  $(G' - b, \mathcal{F}' \cup \{N_{G'}(b)\}, k')$  is a NO-instance, with probability 1. Consequently, if  $(G, \mathcal{F}, k)$  is a NO-instance, then the output is NO or a NO-instance  $(G^*, \mathcal{F}^*, k^*)$  with probability 1.

Let  $(G, \mathcal{F}, k)$  be a YES-instance. By Observation 3.3, after the application of Reduction Rules 3.1 to 3.9, in the reduced instance,  $|\mathcal{F}'| = |\mathcal{F}|$ . Thus, Step 4 is applied at most  $k^2 + 1$  times. Each execution of Step 4 is a *success* with probability at least  $1/(445|\widehat{\mathcal{F}}|^6 + 68)$ , where  $\widehat{\mathcal{F}}$  is the family in the instance considered in that step. In Step 4, the size of the family of any instance is bounded by  $k^2$ , due to Step 2. Hence each execution of Step 4 is a success with probability at least  $1/(445k^{12} + 68)$ . This implies that either our algorithm outputs YES or a YES-instance  $(G^*, \mathcal{F}^*, k^*)$  with probability at least  $(445k^{12} + 68)^{-(k^2+1)}$ . By Observation 3.3, we know that after the application of Reduction Rules 3.1 to 3.9, the parameter  $k'$  in the reduced instance is at most the parameter  $k$  in the original instance. Moreover, if the algorithm outputs an instance, then that will happen in Step 2i and there  $k$  decreases by 1. Thus  $k^* < k$ . This proves the correctness of the algorithm.

By Lemma 13, Step 1 runs in  $\mathcal{O}^*((2k^4)^k)$  time. Observe that, Step 2 runs in polynomial time. All the reduction rules run in polynomial time, and are applied only polynomially many times. Step 4 runs in polynomial time, and we have at most  $k^2 + 1$  iterations. Therefore, the total running time is  $\mathcal{O}^*((2k^4)^k)$ . This completes the proof.  $\blacktriangleleft$

By applying Lemma 14 at most  $k$  times, we can show the following.

**► Lemma 15.** *There exists a randomized algorithm  $\mathcal{B}$  that takes an instance  $(G = (A \uplus B, E), \mathcal{F}, k)$  of HDBFVS as input, runs in  $\mathcal{O}^*((2k^4)^k)$  time, and outputs either YES or NO with the following guarantee. If  $(G, \mathcal{F}, k)$  is a YES-instance, then the output is YES with probability at least  $(445k^{12} + 68)^{-k(k^2+1)}$ . If  $(G, \mathcal{F}, k)$  is a NO-instance, then the output is NO with probability 1.*

Let  $\tau(k) = (445k^{12} + 68)^{k(k^2+1)}$ . To boost the success probability of algorithm  $\mathcal{B}$ , we repeat it  $\mathcal{O}(\tau(k) \log n)$  times. After applying algorithm  $\mathcal{B}$   $\mathcal{O}(\tau(k) \log n)$  times, the success probability is at least  $1 - \left(1 - \frac{1}{\tau(k)}\right)^{\mathcal{O}(\tau(k) \log n)} \geq 1 - \frac{1}{2^{\mathcal{O}(\log n)}} \geq 1 - \frac{1}{n^{\mathcal{O}(1)}}$ .

Thus, we have the following result.

► **Theorem 16.** *There exists a randomized algorithm  $\mathcal{A}$  that takes an instance  $(G = (A \uplus B, E), \mathcal{F}, k)$  of HDBFVS as input, runs in  $\mathcal{O}^*(2^{\mathcal{O}(k^3 \log k)})$  time, and outputs either YES or NO with the following guarantee.*

- If  $(G, \mathcal{F}, k)$  is a YES-instance, then the output is YES with probability at least  $1 - \frac{1}{n^{\mathcal{O}(1)}}$ .
- If  $(G, \mathcal{F}, k)$  is a NO-instance, then the output is NO with probability 1.

## 4 Conclusion and Open Problems

In this paper, we initiated the study of FEEDBACK VERTEX SET problem on hypergraphs. We showed that the problem is W[2]-hard on general hypergraphs. However, when the input is restricted to  $d$ -hypergraphs and linear hypergraphs, which are a strict generalization of graphs, FVS turns out to be tractable (FPT). Derandomization of the randomized FVS algorithm given in this paper is yet to be explored. We believe that this opens up a new direction in the study of parameterized algorithms. That is, extending the study of other graph problems, in the realm of Parameterized Complexity, to hypergraphs. Designing substantially faster algorithms for HFVS on linear hypergraphs and designing polynomial kernels remain interesting questions for the future.

---

### References

- 1 Faisal N. Abu-Khzam. A kernelization algorithm for  $d$ -hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010.
- 2 Akanksha Agrawal, Sushmita Gupta, Saket Saurabh, and Roohani Sharma. Improved algorithms and combinatorial bounds for independent feedback vertex set. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24–26, 2016, Aarhus, Denmark*, volume 63, pages 2:1–2:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
- 3 Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11–15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2016.
- 4 Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous feedback vertex set: A parameterized perspective. *TOCT*, 10(4):18:1–18:25, 2018.
- 5 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res.*, 12:219–234, 2000.
- 6 Yixin Cao. A naive algorithm for feedback vertex set. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7–10, 2018, New Orleans, LA, USA*, volume 61, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 7 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.
- 8 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.
- 9 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40–42):3736–3756, 2010.
- 10 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Trans. Algorithms*, 13(3):43:1–43:22, 2017.

- 11 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.
- 12 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discrete Math.*, 27(1):290–309, 2013.
- 13 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An  $O(2^{O(k)}n^3)$  FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007.
- 14 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.
- 15 Zhuo Diao and Zhongzheng Tang. On the feedback number of 3-uniform hypergraph. *CoRR*, abs/1807.10456, 2018. [arXiv:1807.10456](https://arxiv.org/abs/1807.10456).
- 16 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 2012.
- 17 J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, 2006.
- 18 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 19 Toshihiro Fujito. Approximating minimum feedback vertex sets in hypergraphs. *Theoretical Computer Science*, 246(1):107–116, 2000.
- 20 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.
- 21 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, pages 22:1–22:11, 2019.
- 22 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016.
- 23 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all CSPs, half-integral a-path packing, and linear-time FPT algorithms. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 462–473. IEEE Computer Society, 2018.
- 24 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the s-cycle packing problem. *J. Comb. Theory, Ser. B*, 102(4):1020–1034, 2012.
- 25 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.
- 26 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.
- 27 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size  $k$  in  $O^*(2.7^k)$  time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 971–989, 2020.
- 28 Shaohua Li and Marcin Pilipczuk. An improved FPT algorithm for independent feedback vertex set. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159, pages 344–355. Springer, 2018.
- 29 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. *ACM Trans. Algorithms*, 14(1):7:1–7:37, 2018.
- 30 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theor. Comput. Sci.*, 461:65–75, 2012.

- 31 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *J. Comb. Optim.*, 24(2):131–146, 2012.
- 32 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Trans. Algorithms*, 9(1):11:1–11:23, 2012.
- 33 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theor. Comput. Sci.*, 770:62–68, 2019.
- 34 Junjie Ye. A note on finding dual feedback vertex set. *CoRR*, abs/1510.00773, 2015. [arXiv:1510.00773](https://arxiv.org/abs/1510.00773).



# Size Bounds on Low Depth Circuits for Promise Majority

Joshua Cook

The University Of Texas At Austin, TX, USA

<https://www.cs.utexas.edu/~jacook7/>

[jac22855@utexas.edu](mailto:jac22855@utexas.edu)

---

## Abstract

We give two results on the size of AC0 circuits computing promise majority.  $\epsilon$ -promise majority is majority promised that either at most an  $\epsilon$  fraction of the input bits are 1 or at most  $\epsilon$  are 0.

- First, we show super-quadratic size lower bounds on both monotone and general depth-3 circuits for promise majority.

- For any  $\epsilon \in (0, 1/2)$ , monotone depth-3 AC0 circuits for  $\epsilon$ -promise majority have size

$$\tilde{\Omega}\left(\epsilon^3 n^{2+\frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right).$$

- For any  $\epsilon \in (0, 1/2)$ , general depth-3 AC0 circuits for  $\epsilon$ -promise majority have size

$$\tilde{\Omega}\left(\epsilon^3 n^{2+\frac{\ln(1-\epsilon^2)}{2\ln(\epsilon)}}\right).$$

These are the first quadratic size lower bounds for depth-3  $\epsilon$ -promise majority circuits for  $\epsilon < 0.49$ .

- Second, we give both uniform and non-uniform sub-quadratic size constant-depth circuits for promise majority.

- For integer  $k \geq 1$  and constant  $\epsilon \in (0, 1/2)$ , there exists monotone *non* uniform AC0 circuits of depth- $(2 + 2k)$  computing  $\epsilon$ -promise majority with size

$$\tilde{O}\left(n^{\frac{1}{1-2^{-k}}}\right).$$

- For integer  $k \geq 1$  and constant  $\epsilon \in (0, 1/2)$ , there exists monotone *uniform* AC0 circuit of depth- $(2 + 2k)$  computing  $\epsilon$ -promise majority with size

$$n^{\frac{1}{1-(\frac{2}{3})^k} + o(1)}.$$

These circuits are based on incremental improvements to existing depth-3 circuits for promise majority given by Ajtai [2] and Viola [14] combined with a divide and conquer strategy.

**2012 ACM Subject Classification** Theory of computation → Circuit complexity

**Keywords and phrases** AC0, Approximate Counting, Approximate Majority, Promise Majority, Depth 3 Circuits, Circuit Lower Bound

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.19

**Related Version** A full version of the paper is available at [7], <https://eccc.weizmann.ac.il/report/2020/122/>.

**Funding** This research was supported by NSF grant number 1705028.

**Acknowledgements** Thanks to Dana Moshkovitz for suggesting I study the size cost of derandomizing AC0 circuits. Thanks to Justin Yirka, Amanda Priestly and an anonymous reviewer for feedback on this paper.



© Joshua Cook;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 19; pp. 19:1–19:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

The majority function is a classic function that cannot be computed in AC0 [9]. But AC0 can compute majority promised the input is either mostly 1s or mostly 0s.

► **Definition 1** ( $\epsilon$ -Promise Majority). *Let  $W : \{0, 1\}^n \rightarrow [n]$  be the function giving the number of ones in the input<sup>1</sup>. Let  $\epsilon \in (0, 1/2)$ . Then define the  $\epsilon$  promise inputs to be:*

$$\begin{aligned} \text{Maj}_\epsilon^0 &= \{x \in \{0, 1\}^n : W(x) \leq \epsilon n\} \\ \text{Maj}_\epsilon^1 &= \{x \in \{0, 1\}^n : W(x) \geq (1 - \epsilon)n\} \\ \text{Maj}_\epsilon &= \text{Maj}_\epsilon^0 \cup \text{Maj}_\epsilon^1 \end{aligned}$$

We say that function  $f$  solves the  $\epsilon$ -promise majority<sup>2</sup> problem if:

$$\begin{aligned} f(\text{Maj}_\epsilon^0) &= 0 \\ f(\text{Maj}_\epsilon^1) &= 1 \end{aligned}$$

That is,  $f$  computes the majority promised the input is in  $\text{Maj}_\epsilon$ .

We give size<sup>3</sup> lower bounds to depth-3 circuits<sup>4</sup> computing  $\epsilon$ -promise majority. Then we give small circuits solving promise majority with larger depth.

### 1.1 Motivation

Promise majority is an important tool in derandomizing circuits. We say a function  $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$  is a randomized function for  $g : \{0, 1\}^n \rightarrow \{0, 1\}$  if for all  $x \in \{0, 1\}^n$ ,  $\Pr_{r \in \{0, 1\}^m} [f(x, r) = g(x)] \geq 2/3$ . A circuit implementing  $f$  is called a randomized circuit for  $g$ . We call  $r \in \{0, 1\}^m$  a seed for  $f$ .

Adleman [1] showed that for any randomized function  $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ , implementing some  $g : \{0, 1\}^n \rightarrow \{0, 1\}$ , there is some choice of seeds  $R \subseteq \{0, 1\}^m$  with  $|R| = O(n)$  such that for all  $x$  and the majority of seeds in  $R$ ,  $f$  computes  $g$ , i.e.,  $\Pr_{r \in R} [f(x, r) = g(x)] > 1/2$ . If  $f$  has size- $O(n)$  random circuits, this gives a size- $O(n^2)$  deterministic circuits by computing majority of  $|R|$  copies of  $f$  and taking majority.

Unfortunately, AC0 cannot compute majority, but it can compute  $\epsilon$ -promise majority. With the same argument, we can get  $R$  with  $|R| = O(n)$  such that  $\Pr_{r \in R} [f(x, r) = g(x)] > 3/5$ . So, we only need to compute 2/5-promise majority since  $f$  only outputs the wrong bit for at most 2/5 of  $r \in R$ .

Ajtai [2] gave depth-3 circuits of size  $O\left(n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon) - \ln(1-\epsilon)}}\right)$  solving the  $\epsilon$ -promise majority problem. Applying a depth- $d$  promise majority circuit,  $M$ , to the output of a depth- $k$  circuit,  $C$ , gives a depth- $(k + d - 1)$  circuit since the kind of gate at the lowest level of  $M$  can be made the same as the top level of  $C$ . Combining this result with Adleman takes a size- $O(n)$ , depth- $d$  randomized circuit and gives a depth- $(d + 2)$ , size- $O\left(n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon) - \ln(1-\epsilon)}}\right)$  deterministic circuit. This is bigger than the ideal  $O(n^2)$  size from the unbounded depth setting.

<sup>1</sup> For functions and circuits, we implicitly refer to a family of functions, one for each size  $n$  where  $n$  is implicit. The same holds for  $\text{Maj}_\epsilon$ .

<sup>2</sup> Prior work often called promise majority “approximate majority” [14, 15]. But, approximate majority also refers to the standard notion of approximating a Boolean function [5]. To avoid confusion, we follow the convention suggested in [11] to refer to the promise problem version of majority as promise majority.

<sup>3</sup> In this paper, we use size of a circuit to mean the number of gates.

<sup>4</sup> In this paper, all circuits are constant depth alternating circuits (AC0 circuits) unless stated otherwise.

This paper gives new, super-quadratic size lower bounds for depth-3 circuits computing  $\epsilon$ -promise majority. Thus applying Adleman's technique on AC0 circuits to get size- $O(n^2)$  deterministic circuits using promise majority requires a depth-3 increase. We show this is tight by giving size- $O(n^2)$  depth-4 circuits for  $\epsilon$ -promise majority. Thus Adleman's technique can be used to get size- $O(n^2)$  deterministic circuits with a depth-3 increase.

## 1.2 Our Results

For notation, let  $\tilde{O}(x)$  indicate order  $x$  up to polylogarithmic factors:

- **Definition 2.**  $f(n) = \tilde{O}(g(n))$  if for some integer  $c$ ,  $f = O(g(n) \ln(n)^c)$ .  
 $f(n) = \tilde{\Omega}(g(n))$  if for some integer  $c$ ,  $f = \Omega(g(n) \ln(n)^c)$ .

First, we give a size lower bound for monotone, depth-3 circuits for promise majority. Note that the best known depth-3 circuits are monotone.

- **Theorem 3.** For any  $\epsilon \in (0, 1/2)$ , a monotone, depth-3 circuit solving the  $\epsilon$ -promise majority problem must have size  $\tilde{\Omega}\left(\epsilon^3 n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right)$ .

We follow this up with some weaker, but still super-quadratic, size lower bounds for depth-3 circuits computing promise majority.

- **Theorem 4.** For any  $\epsilon \in (0, 1/2)$ , a depth-3 circuit solving the  $\epsilon$ -promise majority problem must have size  $\tilde{\Omega}\left(\epsilon^3 n^{2 + \frac{\ln(1-\epsilon^2)}{2 \ln(\epsilon)}}\right)$ .

Minor improvements to Ajtai's promise majority circuits [2] gives depth-4, quadratic size, promise majority circuits.

- **Theorem 5.** For constant  $\epsilon \in (0, 1/2)$ , there exists non uniform, monotone, depth-4, size- $O(n^2)$  circuits solving the  $\epsilon$ -promise majority problem.

We then show how to solve  $\epsilon$ -promise majority with even smaller circuits with larger depths using a divide and conquer strategy.

- **Theorem 6.** For constant  $\epsilon \in (0, 1/2)$ , there exists non uniform, monotone, depth- $(2 + 2k)$  circuits solving the  $\epsilon$ -promise majority problem with size  $\tilde{O}\left(n^{\frac{1}{1-2^{-k}}}\right)$ .

The above circuits are not explicit, or uniform: we do not know how to construct it efficiently. However, we next give P-Uniform circuits for promise majority: circuits with a polynomial-time algorithm to construct them. These circuits use a slight improvement to Viola's depth-3 promise majority circuits [14] with a divide and conquer strategy.

- **Theorem 7.** For constant  $\epsilon \in (0, 1/2)$ , there exists P-uniform, monotone, depth- $(2 + 2k)$  circuits solving the  $\epsilon$ -promise majority problem with size  $n^{\frac{1}{1 - (\frac{2}{3})^k} + o(1)}$ .

For  $k = 2$ , this gives depth-6, size- $o(n^2)$ , P-uniform, monotone circuits for promise majority.

- **Corollary 8.** For constant  $\epsilon \in (0, 1/2)$ , there exists P-uniform, monotone, depth-6 circuits solving the  $\epsilon$ -promise majority problem with size  $n^{\frac{9}{5} + o(1)}$ .

Thus a P-uniform PRG with  $O(n)$  seeds for AC0 could derandomize linear-size, randomized circuits to get quadratic-size, deterministic circuits with a depth increase of 5. Finding such PRGs, or even PRGs with polynomially many seeds, is still open. Though, work by Dean Doron, Dana Moshkovitz, Justin Oh and David Zuckerman constructs nearly quadratic PRGs conditioned on some complexity theoretic assumptions [8].

### 1.3 Related Work

There are well known polynomial-size AC0 circuits for promise majority. First, Ajtai gave polynomial size, depth-3 circuits for  $\epsilon$ -promise majority [2]. Ajtai later gave uniform, even deterministic log time uniform, AC0 circuits for promise majority [3]. But, these uniform circuits have large depth and their constructions are complicated. Viola later gave simpler P-Uniform, depth-3 AC0 circuits for promise majority [14].

Chaudhuri and Radhakrishnan [6] proved that any depth- $d$  circuit computing  $\epsilon$ -promise majority must have size  $\Omega\left(\left(\epsilon n\right)^{\frac{1}{1-1/4^d}} - n\right)$ . This gives super-linear lower bounds for depth-3 circuits, but not close to quadratic. Their paper uses deterministic restrictions for lower bounds similar to ours, but our paper uses fan-in lower bounds from Viola [14] and different restrictions to get better depth-3 lower bounds.

In the same work [6], Chaudhuri and Radhakrishnan gave, for any  $k$ , depth- $O(k)$  circuits with size  $O\left(n^{1+\frac{1}{2^k}}\right)$  for  $\epsilon$ -promise majority. Like our paper, it uses a recursive strategy, but we use a different recursive strategy that gives shallower circuits.

Exact threshold functions in AC0 have been studied extensively. Ragde and Wigderson [13] show that for integer  $r > 0$ , the  $\ln(n)^r$  threshold function, which computes whether  $W(x) > \ln(n)^r$ , has AC0 circuits with depth  $O(r)$  and size  $o(n)$ . This improves on a result by Ajtai and Ben-Or [4]. Further, Håstad, Wegener, Wurm, and Yi [10] show that polylogarithmic threshold functions have sub-polynomial size, constant-depth circuits.

Results by Amano [5] building on work by O’Donnell and Wimmer [12] prove the minimum size for a depth- $d$  circuit computing majority on most inputs is  $\Theta\left(n^{\frac{1}{2^d-1}}\right)$ . This is consistent with promise majority results because most inputs are close to balanced, within a  $O(1/\sqrt{n})$  factor, but promise majority is only guaranteed to give majority on inputs that are far from balanced.

For  $\epsilon = \left(\frac{1}{2} - \frac{1}{\ln(n)^k}\right)$ , Viola proved that randomized, depth- $(k+1)$ , polynomial-size circuits can solve  $\epsilon$ -promise majority, but deterministic, depth- $(k+2)$ , polynomial-size circuits cannot. Further, there are deterministic, depth- $(k+3)$ , polynomial-size circuits for  $\epsilon$ -promise majority [15].

The same work [15] gave size lower bounds for depth-3  $\epsilon$ -promise majority circuits, but the bounds are less than linear for  $\epsilon < 0.49$ . Closer analysis gives better lower bounds, but we could not get quadratic lower bounds for  $\epsilon < 0.49$  with this technique.

A later work by Limaye, Srinivasan and Tripathi [11] showed that deterministic, depth- $(k+1)$ , polynomial-size AC0 circuits with parity gates also cannot solve  $\left(\frac{1}{2} - \frac{1}{\ln(n)^k}\right)$ -promise majority.

## 2 Proof Ideas

### 2.1 Monotone Depth-3 Circuit Lower Bounds

For depth-3 promise majority circuits, without loss of generality, assume the first level of gates are AND gates<sup>5</sup>. Call the inputs “variables”, the first level gates “clauses”, and the second level gates “DNFs”.

<sup>5</sup> Switching the ANDs to ORs and ORs to ANDs in a circuit solving  $\epsilon$ -promise majority still solves  $\epsilon$ -promise majority. To see this, observe that flipping all the input bits will flip a  $\text{Maj}_\epsilon^1$  input to a  $\text{Maj}_\epsilon^0$  input. Then apply de Morgan’s law.

To prove lower bounds for a depth-3 circuit, we construct adversarial restrictions that simplify the circuit while setting too few variables to violate the promise. To do this, we use two main tools. The first is a lemma from Viola [14] that we use to remove gates with very small fan in at the first level.

The second is a greedy set cover algorithm which shows that any collections of large subsets of variables can have a large fraction of the subsets hit by a small fraction of variables. To do this, we repeatedly select a variable in at least the average number of sets per variable.

First, we show DNFs have  $\tilde{\Omega}(n^{1+\alpha})$  clauses for some  $\alpha > 0$ . To do this, we eliminate small clauses using the first idea, then eliminate a large fraction of clauses with few 0s using the second idea. This leaves many clauses while eliminating a large fraction of clauses, thus we started with many clauses.

Then, we show the circuit has  $\tilde{\Omega}(n^{2+\alpha})$  clauses. First, we use the second idea to remove any very large clauses. This lets us fix clauses to 1 without using too many variables. Then, using the second idea again, we can hit many DNFs using few clauses. Thus there must be many clauses so we can not hit every DNF using few clauses.

We generally will not worry about integrality. This only becomes an issue when  $\epsilon = \tilde{O}(n^{-1/2})$  as some restrictions would not have size greater than one. In that case, our lower bounds hold trivially as  $\epsilon n$  gates can be fixed to a constant assigning only  $\epsilon n$  variables.

## 2.2 General Depth-3 Circuit Lower Bounds

The proof for non-monotone circuits is similar but with an additional hurdle. In monotone circuits, setting variables to 0 only makes clauses 0. But with negations, we can actually shrink clauses without eliminating them. This is an issue for showing DNFs must be large, but the rest of the argument only needs minor changes.

The solution is to set adversarial bits probabilistically. We independently set each bit to 1 with probability  $\epsilon$ . With good probability, this will give an input in  $\text{Maj}_\epsilon^0$ . Some DNFs then must have a good probability of “noticing” and becoming 0.

With high probability, fixing a small fraction of variables according to  $D_\epsilon$  will eliminate many clauses. For some  $\alpha > 0$ , if a DNF is smaller than  $n^{1+\alpha}$  this will make it constant. With good probability, setting the rest of the variables gives an input this DNF must “notice” and become 0. Thus, if the DNF is small, for some input it will be fixed to the constant 0 with only a few variables fixed. This cannot happen, so the DNF must be larger than  $n^{1+\alpha}$ .

## 2.3 Small Sized Circuits

To get small circuits, first we amplify the  $\epsilon$  promise input to a  $\frac{1}{\text{polylog}(n)}$  promise input by taking majority over  $O(\ln(\ln(n)))$  length walks on an expander graph. Then we separate our input into polynomially small groups and run a  $\frac{1}{\ln(n)}$ -promise majority on each. This gives a polynomially smaller layer which satisfies just an  $\ln(n)$  factor worse promise. Applying this several times computes majority of the promise input.

Ajtai’s promise majority strategy gives a quadratic-sized  $\frac{1}{\ln(n)}$ -promise majority circuit. Using this with the divide and conquer strategy above gives non uniform small circuits.

For our uniform circuit, we look at Viola’s circuit [14]. It uses a hitting property that requires  $n^{3+o(1)}$  many random walks for each of our  $n$  bits, requiring an overall size of  $n^{4+o(1)}$ . We reduce this by showing it suffices to let each bit only range over random walks starting at that bit, giving a size- $n^{3+o(1)}$  circuit for  $\frac{1}{\ln(n)}$ -promise majority.

Applying this improved version of Viola’s depth-3 circuit with our divide and conquer strategy gives our uniform small circuits.

## 2.4 Terminology

We will use biased inputs in our proofs.

► **Definition 9** ( $\epsilon$  Biased Input). For any  $\epsilon \in [0, 1]$  the  $\epsilon$  biased input  $D_\epsilon$  is a random variable over  $\{0, 1\}^n$  where each bit independently is 1 with probability  $\epsilon$ .

As with  $\text{Maj}_\epsilon^0$  and  $\text{Maj}_\epsilon^1$ ,  $n$  in  $D_\epsilon$  is implicit.  $D_\epsilon$  is related to  $\text{Maj}_\epsilon^0$  by a central limit theorem:  $\Pr[D_\epsilon \in \text{Maj}_\epsilon^0] > \frac{1}{3}$  for large enough  $n$ .

We will make sub DNFs by only taking some clauses from a larger DNF.

► **Definition 10** (Sub DNF). Let  $G$  be a DNF with clauses  $C = \{C_i : i \in [k]\}$  so that  $G = \bigvee_{i \in [k]} C_i$ . Let  $\Lambda \subseteq [k]$  and  $H$  be a DNF with  $H = \bigvee_{i \in \Lambda} C_i$ .

Then we say that  $H$  is sub DNF of  $G$  or  $G$  has sub DNF  $H$ .

Restrictions fix some bits in the input to a function. We formalize this as a function that takes unrestricted bits as input and outputs the restricted and unrestricted bits together<sup>6</sup>.

► **Definition 11** (Restriction). A restriction  $\rho$  on  $n$  variables of size  $m$  is a function  $\rho : \{0, 1\}^{n-m} \rightarrow \{0, 1\}^n$  such that for some  $c \in \{0, 1\}^m$  and some permutation of  $[n]$ ,  $\pi$ , for all  $x \in \{0, 1\}^{n-m}$  and  $i \in n$ :

$$\rho(x)_i = \begin{cases} c_{\pi_i} & \pi_i \leq m \\ x_{\pi_i - m} & \pi_i > m \end{cases}$$

We write the size of  $\rho$  as  $|\rho| = m$  and define  $f \upharpoonright_\rho = f \circ \rho$ .

When we apply a restriction,  $\rho$ , to a DNF,  $F$ , we let  $F \upharpoonright_\rho$  be the DNF which is  $F$  with variables restricted in  $\rho$  set to their restricted value. We simplify such a DNF to remove any clause that has been set to 0. We count the size of a DNF by its number of clauses.

► **Definition 12** (DNF Size and Width). For a DNF  $F$ , the size of  $F$ ,  $|F|$ , is the number of clauses in  $F$ . Any DNF that is the constant 1 or 0 function has size 0.

We say a DNF  $F$  has width  $w$  if no clause in  $F$  has width greater than  $w$ .

## 3 Monotone Depth-3 Circuit Size Lower Bounds

### 3.1 Removing Small Clauses

We use a result from Viola [14], Lemma 11 therein. Intuitively, this lemma says for a DNF with small width, either there is some setting to a small number of variables that makes it 0, or under a randomized input it is unlikely to be 0.

► **Lemma 13.** Let  $G$  be a DNF with a sub DNF  $F$ . Assume for some positive integers  $w$  and  $m$ ,  $F$  has width at most  $w$  and  $\Pr[G(D_\epsilon) = 0] \geq e^{-\epsilon^w \cdot m/w^2}$ . Then there exists a restriction  $\rho$  with  $|\rho| \leq m$  such that  $F \upharpoonright_\rho = 0$  and  $\Pr[G \upharpoonright_\rho (D_\epsilon) = 0] \geq \Pr[G(D_\epsilon) = 0]$ .

Our result is slightly generalized over the original. See the full version of this paper for details. As a corollary, we can apply small restrictions to eliminate small width clauses.

<sup>6</sup> This is an equivalent but slightly nonstandard way to define restrictions.

► **Corollary 14.** *Suppose we have  $\epsilon \in (0, 1/2)$ , DNF  $F$  and constant  $\alpha > 0$  such that  $\Pr[F(D_\epsilon) = 0] \geq n^{-\alpha}$ . Then for sufficiently large  $n$  and*

$$w = \log_\epsilon \left( \frac{\ln(n)^5}{n\epsilon \ln(\epsilon)^2} \right)$$

*there is a restriction  $\rho$  restricting at most  $m = \frac{\epsilon n}{\ln(n)}$  variables so that any clause  $C$  in  $F$  with width less than  $w$  has  $C \upharpoonright_\rho = 0$  and  $\Pr[F \upharpoonright_\rho (D_\epsilon) = 0] \geq \Pr[F(D_\epsilon) = 0]$ .*

**Proof.** Let  $F'$  be the sub DNF of  $F$  with clauses of width less than  $w$ . Then

$$\mathbb{E}[F(D_\epsilon) = 0] \geq n^{-\alpha} = e^{-\alpha \ln(n)} = e^{-\alpha \frac{\ln(n)^5}{n\epsilon \ln(\epsilon)^2} \frac{\epsilon n}{\ln(n)} \frac{\ln(\epsilon)^2}{\ln(n)^3}} = e^{-\alpha \epsilon^w m \frac{\ln(\epsilon)^2}{\ln(n)^3}} \geq e^{-\epsilon^w m \frac{1}{w^2}}$$

From Lemma 13, there is a restriction  $\rho$  of size  $m$  with  $\mathbb{E}[F \upharpoonright_\rho (D_\epsilon) = 0] \geq \mathbb{E}[F(D_\epsilon) = 0]$  setting  $F' \upharpoonright_\rho = 0$ . Any width  $w$  clause  $C$  would be in  $F'$ , thus  $C \upharpoonright_\rho = 0$  since  $F' \upharpoonright_\rho = 0$ . ◀

### 3.2 Covering Many Large Sets with Few Elements

We prove the simplest version of the clause elimination result, but slight variations will be used in multiple places. In particular, in the non-monotone lower bounds, we can't quite reduce the problem to set cover, but the same algorithm still works with a similar bound. Since the proofs look very similar, we only present one in detail. We show how to remove many clauses from a monotone DNF with a small restriction

► **Lemma 15.** *Let  $F$  be a monotone DNF where each clause has width at least  $w$ . Then for any positive integer  $b$ , there is some restriction  $\rho$  with  $|\rho| = b$  only fixing variables to 0 such that  $|F \upharpoonright_\rho| < |F|e^{w \ln(1 - \frac{b}{n+1})}$*

**Proof.** The idea is to restrict the variable that intersects the most clauses to 0. This removes at least the average number of clauses per variable, which when we have  $m$  clauses and have fixed  $i$  variables is at least  $\frac{mw}{n-i}$ . After  $b$  restrictions, we get  $\rho$  with  $|\rho| = b$  and

$$|F \upharpoonright_\rho| \leq |F| \prod_{i=0}^{b-1} \left( 1 - \frac{w}{n-i} \right).$$

We prove this by induction then simplify. For the base case where  $b = 0$ ,  $F$  is unchanged and we get the empty product, so the inequality holds.

For  $b > 0$ , we have some  $\rho'$  restricting  $b-1$  variables with  $|F \upharpoonright_{\rho'}| \leq |F| \prod_{i=0}^{b-2} \left( 1 - \frac{w}{n-i} \right)$ . Then  $F \upharpoonright_{\rho'}$  is a function on  $n+1-b$  variables. Let  $s$  be the variable in the most clauses of  $F \upharpoonright_{\rho'}$ . Then  $s$  is in at least  $|F \upharpoonright_{\rho'}| \frac{w}{n+1-b}$  clauses. Let  $\rho$  be  $\rho'$  also fixing  $s$  to 0. Then:

$$|F \upharpoonright_\rho| \leq |F \upharpoonright_{\rho'}| - |F \upharpoonright_{\rho'}| \frac{w}{n+1-b} = |F| \prod_{i=0}^{b-1} \left( 1 - \frac{w}{n-i} \right),$$

completing our induction. The above equation simplifies to:

$$|F \upharpoonright_\rho| = |F| \prod_{i=0}^{b-1} \left( 1 - \frac{w}{n-i} \right) < |F| e^{\sum_{i=0}^{b-1} -\frac{w}{n-i}} = |F| e^{-w \sum_{i=n+1-b}^n 1/i}. \quad (1)$$

From calculus we have

$$\sum_{i=a}^b \frac{1}{i} \geq \int_a^{b+1} \frac{1}{x} dx = \ln \left( \frac{b+1}{a} \right),$$

## 19:8 Size Bounds on Low Depth Circuits for Promise Majority

which applied to Equation (1) gives

$$|F \upharpoonright_\rho| < |F|e^{-w \sum_{i=n+1-b}^n \frac{1}{i}} \leq |F|e^{-w \ln\left(\frac{n+1}{n+1-b}\right)} = |F|e^{w \ln\left(1 - \frac{b}{n+1}\right)}. \quad \blacktriangleleft$$

The same idea gives the simpler bound:

► **Corollary 16.** *Let  $F$  be a monotone DNF where each clause has width at least  $w$ . Then for any integer  $b$  there is some restriction  $\rho$  with  $|\rho| \leq b$  such that  $|F \upharpoonright_\rho| < |F|e^{-wb/n}$ .*

With this idea, we can remove all large clauses fixing few variables. For the non-monotone case, we only remove half the average number of clauses with each variable, giving:

► **Corollary 17.** *Let  $F$  be a collection of clauses. Then there is some restriction  $\rho$  fixing  $n/p$  variables such that  $F \upharpoonright_\rho$  has width  $w = 2 \ln(|F|p)$ .*

### 3.3 Monotone DNF Size

We prove that any DNF with a good chance of “noticing” inputs from  $D_\epsilon$  has a large size.

► **Lemma 18.** *Suppose for  $\epsilon \in (0, 1/2)$ , there is a monotone DNF  $F$  with  $F(\text{Maj}_\epsilon^1) = 1$  and  $\Pr[F(D_\epsilon) = 0] \geq 1/n^\alpha$  for constant  $\alpha$ . Then  $F$  has  $\tilde{\Omega}\left(\epsilon n^{1 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}$  clauses.*

**Proof.** The idea is to restrict our function until we are only promised it outputs 1 on an  $\text{Maj}_{\epsilon/\ln(n)}^1$  input. Using Lemma 15, we can do this in such a way that eliminates a large fraction of clauses. Then since we still need to output 1 if we have fewer than  $\frac{\epsilon n}{\ln(n)}$  more zeros, we can choose these remaining  $\frac{\epsilon n}{\ln(n)}$  zeros to each eliminate one clause, showing that there are still  $\frac{\epsilon n}{\ln(n)}$  clauses left. This will imply that we must have started with the claimed number of clauses.

For  $w = \log_\epsilon\left(\frac{\ln(n)^5}{n\epsilon \ln(\epsilon)^2}\right)$ , by Corollary 14, there is restriction  $\rho$  with  $|\rho| \leq \frac{\epsilon n}{\ln(n)}$  and  $F \upharpoonright_\rho$  that has no clauses smaller than  $w$ . Denote  $F_2 = F \upharpoonright_\rho$ . Note  $F_2$  solves  $F_2\left(\text{Maj}_{\epsilon(1-1/\ln(n))}^1\right) = 1$  and has no clauses smaller than  $w$ .

Now we use Lemma 15 to get restriction  $\rho_2$  that assigns  $\epsilon n(1 - 2/\ln(n))$  variables and:

$$|F_2 \upharpoonright_{\rho_2}| \leq |F_2|e^{w \ln\left(1 - \frac{\epsilon n(1-2/\ln(n))}{n+1}\right)}.$$

Now we simplify the above exponent. For  $0 < x < \frac{1}{2}$  and  $0 < y$ , by a Taylor argument we have  $\ln(1 - x + y) \leq \ln(1 - x) + 2y$ . Then for sufficiently large  $n$ :

$$\ln\left(1 - \frac{\epsilon n(1 - 2/\ln(n))}{n+1}\right) = \ln\left(1 - \epsilon + \frac{\epsilon}{n+1} + \frac{2\epsilon n}{n+1} \frac{1}{\ln(n)}\right) \leq \ln(1 - \epsilon) + \frac{5\epsilon}{\ln(n)}.$$

Now including  $w$ ,

$$\begin{aligned} w \ln\left(1 - \frac{\epsilon n(1 - 2/\ln(n))}{n+1}\right) &\leq \frac{\ln(n) - \ln\left(\frac{\ln(n)^5}{\epsilon \ln(\epsilon)^2}\right)}{\ln(1/\epsilon)} \left(\ln(1 - \epsilon) + \frac{5\epsilon}{\ln(n)}\right) \\ &< \frac{\ln(n) \ln(1 - \epsilon)}{\ln(1/\epsilon)} - \frac{\ln\left(\frac{\ln(n)^5}{\epsilon \ln(\epsilon)^2}\right) \ln(1 - \epsilon)}{\ln(1/\epsilon)} + \frac{5}{\ln(1/\epsilon)}. \end{aligned}$$

Then applying this to our size bound

$$\begin{aligned} |F_2 \upharpoonright_{\rho_2}| &\leq |F_2| e^{w \ln\left(1 - \frac{\epsilon n(1-2/\ln(n))}{n+1}\right)} \\ &< |F_2| e^{\frac{\ln(n) \ln(1-\epsilon)}{\ln(1/\epsilon)} - \frac{\ln\left(\frac{\ln(n)^5}{\epsilon \ln(\epsilon)^2}\right) \ln(1-\epsilon)}{\ln(1/\epsilon)} + \frac{5}{\ln(1/\epsilon)}} \\ &< |F_2| n^{\frac{\ln(1-\epsilon)}{\ln(1/\epsilon)}} 2 \ln(n)^5 e^8. \end{aligned}$$

Since  $\rho$  and  $\rho_2$  only restricts  $\epsilon n \left(1 - \frac{1}{\ln(n)}\right)$  clauses,  $F_2 \upharpoonright_{\rho_2} (\text{Maj}_{\epsilon/\ln(n)}^1) = 1$ . Further, since  $F$  is monotone,  $\rho$  and  $\rho_2$  only fixed variables to 0. Therefore,  $F_2 \upharpoonright_{\rho_2} \neq 1$ . Then  $F_2 \upharpoonright_{\rho_2}$  must have at least  $\frac{\epsilon n}{\ln(n)}$  clauses. Thus:

$$\begin{aligned} \frac{\epsilon n}{\ln(n)} &\leq |F_2 \upharpoonright_{\rho_2}| \leq e^8 |F_2| n^{\frac{\ln(1-\epsilon)}{\ln(1/\epsilon)}} 2 \ln(n)^5 \\ \frac{\epsilon n^{1 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}}{2e^8 \ln(n)^6} &\leq |F_2|. \end{aligned}$$

$F$  has at least as many clauses as  $F_2$ , thus  $|F| = \tilde{\Omega}\left(\epsilon n^{1 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right)$ . ◀

### 3.4 Monotone Circuit Size Lower Bounds

Now we prove the monotone depth-3 promise majority circuit lower bounds.

► **Theorem 3.** *For any  $\epsilon \in (0, 1/2)$ , a monotone, depth-3 circuit solving the  $\epsilon$ -promise majority problem must have size  $\tilde{\Omega}\left(\epsilon^3 n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right)$ .*

**Proof.** Let  $F$  be a monotone depth-3 circuit computing  $\epsilon$ -promise majority. We will refer to the first level gates as clauses, and the second level gates as DNFs. Let  $|F|$  refer to the number of clauses in  $F$ , and  $\|F\|$  refer to the number of DNFs. If  $F$  has more than  $n^{2+\alpha}$  gates, we are done. So suppose it does not.

Let  $\alpha = \frac{\ln(1-\epsilon+3\epsilon/\ln(n))}{\ln(\epsilon-3\epsilon/\ln(n))}$ . We can show that

$$\alpha > \frac{\ln(1-\epsilon)}{\ln(\epsilon)} - O\left(\frac{1}{\ln(n)}\right).$$

So if we show  $|F| = \tilde{\Omega}\left(\epsilon^3 n^{2+\alpha}\right)$ , then the second term in  $\alpha$  becomes a constant.

First, from Corollary 17, we have a restriction  $\rho$  fixing  $\frac{\epsilon n}{\ln(n)}$  variables such that any clause wider than  $w = 2 \ln(|F|) \frac{\ln(n)}{\epsilon}$  is set to 0. Let  $F_2 = F \upharpoonright_{\rho}$ . See that  $F_2$  solves the  $\epsilon \left(1 - \frac{1}{\ln(n)}\right)$ -promise majority problem and has no clauses wider than  $6 \frac{\ln(n)^2}{\epsilon}$ .

By Lemma 18, every DNF  $G$  with  $\Pr[G(D_{\epsilon(1-3/\ln(n))}) = 0] \geq 1/n^{3+\alpha}$  has at least  $c\epsilon n^{1+\alpha}$  clauses for some polylogarithm  $c$ . Let  $F_3$  be the sub circuit of  $F_2$  with only the DNFs of  $F_2$  larger than  $c\epsilon n^{1+\alpha}$ .

Since no clauses are wider than  $w$ , we can set any  $m$  clauses in  $F_3$  to 0 by fixing only  $mw$  variables. Then, analogous to Corollary 16, there exists a restriction  $\rho_2$  fixing  $\epsilon n/\ln(n)$  variables to 1 such that:

$$\|F_3 \upharpoonright_{\rho_2}\| \leq \|F_3\| e^{-c\epsilon n^{1+\alpha} (|\rho_2|/w)/|F_3|},$$

where  $\|F_3 \upharpoonright_{\rho_2}\|$  is the number of DNFs in  $F_3$  not fixed to 1 or 0 under the restriction  $\rho_2$ .

## 19:10 Size Bounds on Low Depth Circuits for Promise Majority

See that  $F_2 \upharpoonright_{\rho_2}$  still solves the  $\epsilon(1-2/\ln(n))$ -majority problem. By a central limit theorem,  $D_{\epsilon(1-2/\ln(n))}$  has a constant nonzero probability of being in  $\text{Maj}_{\epsilon(1-2/\ln(n))}^0$ . Since  $F_2$  has fewer than  $n^{2+\alpha}$  DNFs (by assumption), some DNF in  $F_2$ ,  $A$ , must be 0 on  $D_{\epsilon(1-2/\ln(n))}$  with probability greater than  $1/n^{3+\alpha}$ . By Lemma 18,  $A$  has size at least  $c\epsilon n^{1+\alpha}$ . Thus  $A$  must also be in  $F_3$ . Thus  $\|F_3 \upharpoonright_{\rho_2}\| \geq 1$ .

Now we can compute a lower bound for  $|F_3|$ :

$$\begin{aligned} 1 &\leq \|F_3 \upharpoonright_{\rho_2}\| \leq \|F_3\| e^{-c\epsilon n^{1+\alpha} \frac{|\rho_2|}{w|F_3|}} \\ e^{c\epsilon^2 n^{2+\alpha} \frac{1}{w|F_3| \ln(n)}} &\leq \|F_3\| \\ c\epsilon^3 n^{2+\alpha} \frac{1}{2 \ln(|F|) \ln(n) |F_3| \ln(n)} &\leq \ln(\|F_3\|) \\ \tilde{\Omega}(\epsilon^3 n^{2+\alpha}) &\leq |F_3|. \end{aligned}$$

Using the definition of  $\alpha$  and that  $|F| > |F_3|$  we get:

$$|F| \geq \tilde{\Omega}(\epsilon^3 n^{2+\alpha}) \geq \tilde{\Omega}\left(\epsilon^3 n^{2+\frac{\ln(1-\epsilon)}{\ln(\epsilon)}} - O\left(\frac{1}{\ln(n)}\right)\right) \geq \tilde{\Omega}\left(\epsilon^3 n^{2+\frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right). \quad \blacktriangleleft$$

### 4 General Depth-3 Circuits

The proof of the size lower bound for general depth-3 circuits computing promise majority is almost the same as the monotone case, except for the proof that DNFs must be large. We only prove our DNF size lower bound here, the circuit lower bound follows similar to the proof of Theorem 3.

► **Lemma 19.** *Suppose  $\epsilon \in (0, 1/2)$ , and  $F$  is a DNF such that  $\Pr[F(D_\epsilon) = 0] \geq 1/n^\alpha$  for some constant  $\alpha$  and  $F(\text{Maj}_\epsilon^1) = 1$ . Then  $F$  has size at least  $\tilde{\Omega}\left(\epsilon n^{1+\frac{\ln(1-\epsilon^2)}{2\ln(\epsilon)}}\right)$ .*

**Proof.** First, see that if  $F(\text{Maj}_\epsilon^1) = 1$  and  $F \neq 1$ , there must be at least  $\epsilon n$  clauses. Otherwise we could fix one variable in each clause to 0 using fewer than  $n\epsilon$  zeros. Then for  $\epsilon = \tilde{O}\left(\frac{1}{\sqrt{n}}\right)$  the lemma is satisfied. So take  $\epsilon = \omega\left(\frac{\ln(n)^3}{\sqrt{n}}\right)$ .

Let  $m = \epsilon n(1-2/\ln(n))$  and  $w = \log_\epsilon\left(\frac{\ln(n)^5}{n\epsilon \ln(\epsilon)^2}\right)$ . We will define a sequence of probabilistic restrictions,  $\rho_0, \dots, \rho_m$ , each restricting one more variable according to  $D_\epsilon$ . At the same time we will construct a sequence of sub DNFs of  $F$ ,  $F_0, \dots, F_m$ , each a subset of the last, so that each  $F_i \upharpoonright_{\rho_i}$  has width at least  $w$ .

Informally, with decent probability each  $F_i$  is significantly smaller than the last. Thus by a Chernoff bound, with high probability  $F_m$  has a small fraction of the clauses of  $F$ . Then we use Corollary 14 to eliminate the small width clauses in  $F_m \upharpoonright_{\rho_m}$ . With good probability the DNF will still not be 1, in which case it must still have an almost linear number of clauses. Thus there must have been many clauses to destroy so many and have so many left.

Let  $\rho_0$  restrict no variables and  $F_0$  be  $F$  restricted to clauses wider than  $w$ . Then for any  $i$ , let  $\rho_i$  be  $\rho_{i-1}$  plus restricting whichever variable appears in the most clauses in  $F_{i-1} \upharpoonright_{\rho_{i-1}}$  to one with probability  $\epsilon$  and 0 otherwise. Then let  $F_i$  be the clauses such that they have width greater than  $w$  in  $F \upharpoonright_{\rho_i}$ . See that  $F_i \subseteq F_{i-1}$ , since further restrictions will only decrease the size and number of clauses.

With probability at least  $\epsilon$ ,  $\rho_i$  will eliminate at least  $\frac{|F_{i-1}|w}{2(n-i+1)}$  clauses. Thus:

$$\Pr\left[|F_{i+1}| \leq |F_i| \left(1 - \frac{w}{2(n-i)}\right)\right] \geq \epsilon.$$

Let  $k$  be the number of times the above inequality holds. By an argument similar to Lemma 15:

$$|F_m| \leq |F_0| \prod_{i=0}^{k-1} \left(1 - \frac{w}{2(n-i)}\right)^k \leq |F_0| e^{\frac{w}{2} \ln\left(1 - \frac{k}{n+1}\right)}.$$

See the expected value of  $k$  is at least  $m\epsilon$ . By a Chernoff bound, we have:

$$\Pr[k < (1 - \frac{1}{\ln(n)})\epsilon m] \leq e^{-\frac{\epsilon m}{2 \ln(n)^2}} < e^{-\frac{\epsilon^2 n}{\ln(n)^3}}.$$

Now, notice that  $\rho_m$  only sets variables according to an  $\epsilon$  biased distribution. So if we just finish sampling the rest of the variables from  $D_\epsilon$ , it is the same as sampling all the variables from  $D_\epsilon$ . Thus:

$$\mathbb{E}_{\rho_m}[\Pr[F \upharpoonright_{\rho_m}(D_\epsilon) = 0]] = \Pr[F(D_\epsilon) = 0].$$

We need high probability that  $F \upharpoonright_{\rho_m}$  still outputs 0 with polynomial probability on  $D_\epsilon$ . Applying the above equation and our assumption we get:

$$\begin{aligned} \frac{1}{n^\alpha} &\leq \mathbb{E}_{\rho_m}[\Pr[F \upharpoonright_{\rho_m}(D_\epsilon) = 0]] \\ &\leq \frac{1}{n^{2\alpha}} + \Pr_{\rho_m}[\Pr[F \upharpoonright_{\rho_m}(D_\epsilon) = 0] > \frac{1}{n^{2\alpha}}] \\ \frac{1}{n^\alpha} - \frac{1}{n^{2\alpha}} &\leq \Pr_{\rho_m}[\Pr[F \upharpoonright_{\rho_m}(D_\epsilon) = 0] > \frac{1}{n^{2\alpha}}]. \end{aligned}$$

The probability that  $\rho_m$  has both  $\Pr[F \upharpoonright_{\rho_m}(D_\epsilon) = 0] > 1/n^{2\alpha}$  and  $k > (1 - \frac{1}{\ln(n)})\epsilon m$  is at least  $\frac{1}{n^\alpha} - \frac{1}{n^{2\alpha}} - e^{-\frac{\epsilon^2 n}{\ln(n)^3}}$ , which for large  $n$  is positive. Then take such  $\rho_m$  as  $\rho$ .

By Corollary 14, we have a restriction of  $F|_{\rho, \rho'}$ , which restricts  $\epsilon n / \ln(n)$  variables and leaves no clauses of width less than  $w$ , and has

$$\Pr[F \upharpoonright_{\rho \upharpoonright_{\rho'}}(D_\epsilon) = 0] \geq \Pr[F \upharpoonright_{\rho}(D_\epsilon) = 0] \geq \frac{1}{n^{2\alpha}}.$$

Now call  $F' = F \upharpoonright_{\rho \upharpoonright_{\rho'}}$ . See that  $F'$  has fixed  $\epsilon n(1 - \frac{1}{\ln(n)})$  variables. Thus it still satisfies  $F'(\text{Maj}_{\epsilon/\ln(n)}^1) = 1$ . Since  $F' \neq 1$ ,  $|F'| \geq \epsilon n / \ln(n)$ . The clauses in  $F'$  had width greater than  $w$  in  $F_m$ , otherwise  $\rho'$  would have set them to 0. Thus  $|F_m| \geq \epsilon n / \ln(n)$ . Together we have:

$$\begin{aligned} \frac{\epsilon n}{\ln(n)} &\leq |F_0| e^{\frac{w}{2} \ln\left(1 - \frac{k}{n+1}\right)} \\ &\leq |F_0| e^{\frac{w}{2} \ln\left(1 - \frac{(1-1/\ln(n))\epsilon m}{n+1}\right)} \\ &\leq |F_0| e^{\frac{\ln\left(\frac{n\epsilon \ln(\epsilon)^2}{\ln(n)^5}\right)}{2 \ln(1/\epsilon)}} (\ln(1-\epsilon^2) + 6\epsilon^2 / \ln(n)) \\ \tilde{\Omega}\left(\epsilon n^{1 + \frac{\ln(1-\epsilon^2)}{2 \ln(\epsilon)}}\right) &\leq |F_0|. \end{aligned}$$

Thus  $F$  has at least  $\tilde{\Omega}\left(\epsilon n^{1 + \frac{\ln(1-\epsilon^2)}{2 \ln(\epsilon)}}\right)$  clauses. ◀

## 5 Circuit Upper Bounds

This section mostly uses standard techniques and the details are left for the full paper. A close analysis of Ajtai's [2] promise majority circuits gives:

► **Theorem 20.** *For any  $\epsilon \in (0, 1/2)$ , there exists monotone, depth-3 circuits solving the  $\epsilon$ -promise majority problem with size  $O\left((\epsilon \ln(\epsilon))^2 n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon) - \ln(1-\epsilon)}}\right)$ .*

This also gives the corollary we will use for our stronger upper bounds for higher depth.

► **Corollary 21.** *For any  $\epsilon = O\left(\frac{\ln(\ln(n))}{\ln(n)}\right)$ , there are monotone, depth-3 circuits solving the  $\epsilon$ -promise majority problem with size  $O(n^2)$ .*

Using random walks on expander graphs, we can amplify our promise. The polylogarithmic factor in the size depends on  $\epsilon$  and  $k$ .

► **Lemma 22.** *For any constant  $k$  and  $\epsilon \in (0, 1/2)$ , there exists  $P$ -Uniform, monotone, depth-3 circuits with size  $\tilde{O}(n)$  amplifying a  $\text{Maj}_\epsilon^0$  input to a  $\text{Maj}_{\frac{1}{\ln(n)^k}}^0$  output and a  $\text{Maj}_\epsilon^1$  input to a  $\text{Maj}_{\frac{1}{\ln(n)^k}}^1$  output.*

With amplification and quadratic-size circuits, we can trivially prove the existence of depth-4, size- $\tilde{O}(n^2)$  circuits for promise majority. But the circuit size only depends on the number of potential inputs (*not* the number of bits used to represent them). Thus the circuit has size  $O(n^2)$ .

► **Theorem 5.** *For constant  $\epsilon \in (0, 1/2)$ , there exists non uniform, monotone, depth-4, size- $O(n^2)$  circuits solving the  $\epsilon$ -promise majority problem.*

We can apply promise majority circuits in a divide and conquer fashion to get the following:

► **Lemma 23.** *If there are depth-3 circuits with size  $n^\alpha$  solving  $\frac{1}{\ln(n)}$ -promise majority, then for any positive integer  $k$ , there are depth- $(1 + 2k)$  circuits solving  $\frac{1}{\ln(n)^k}$ -promise majority with size*

$$kn^{\frac{1}{1 - \left(\frac{\alpha-1}{\alpha}\right)^k}},$$

*which is uniform and monotone if the depth-3 circuits are uniform and monotone.*

Combining Lemma 23 with amplification and our quadratic-sized majority gives:

► **Theorem 6.** *For constant  $\epsilon \in (0, 1/2)$ , there exists non uniform, monotone, depth- $(2 + 2k)$  circuits solving the  $\epsilon$ -promise majority problem with size  $\tilde{O}\left(n^{\frac{1}{1-2^{-k}}}\right)$ .*

For uniform circuits, we refine Viola's result [14] by giving a more efficient way to use the random walks in the existing algorithm.

► **Theorem 24.** *There exists  $P$ -uniform, monotone, depth-3, size- $O(n^{3+o(1)})$  circuits solving the  $\frac{1}{\ln(n)}$ -promise majority problem.*

Again applying amplification and divide and conquer we get:

► **Theorem 7.** *For constant  $\epsilon \in (0, 1/2)$ , there exists  $P$ -uniform, monotone, depth- $(2 + 2k)$  circuits solving the  $\epsilon$ -promise majority problem with size  $n^{\frac{1}{1 - \left(\frac{2}{3}\right)^k} + o(1)}$ .*

## 6 Closing Statements & Open Problems

Some technical details, especially the upper bounds, have been left to the full version of the paper [7] on ECCC at <https://eccc.weizmann.ac.il/report/2020/122/>.

These results are essentially tight in the following sense. For a wide range of  $\epsilon$ , between  $\epsilon = o(1)$  and  $\epsilon = n^{-o(1)}$ , the optimal size of depth-3 circuits for  $\epsilon$ -promise majority is  $n^{2 \pm o(1)}$ .

These lower bounds do not obviously extend to depth-4 circuits, so the right size for promise majority at higher depths is less clear. Better amplification plus Ajtai's promise majority circuit can actually achieve circuits with size significantly smaller than  $n^2$ . So our upper bounds are not optimal for depths greater than 3.

For depth-3 circuits computing promise majority, we gave four different size bounds: a lower bound for monotone circuits, a lower bound for general circuits, an upper bound for monotone circuits, and an upper bound for uniform monotone circuits. Each of these bounds differs by a polynomial factor, but we suspect they are equal.

Finally, here are some open problems:

1. Is there a way to derandomize any depth- $d$ , size- $O(n)$ , randomized circuit to get a depth- $(d+2)$ , size- $O(n^2)$ , deterministic circuit?

We did not find any function  $f$  that has a randomized, depth- $d$ , size- $O(n)$  circuit,  $R$ , computing  $f$ , but no deterministic, depth- $(d+2)$ , size- $O(n^2)$  circuit computing  $f$ . We only showed that taking promise majority over  $O(n)$  copies of  $R$  (as you would with an ideal PRG) would give super-quadratic circuits. There may always be some other deterministic, depth- $(d+2)$ , size- $O(n^2)$  circuit computing  $f$ .

2. Do negations help solve promise majority?

Our lower bounds for monotone circuits are better than our general lower bounds. It does not seem like negations should help, but we were unable to rule it out.

3. What is optimal size for depth-3 circuits computing  $\epsilon$ -promise majority?

For constant  $\epsilon \in (0, 1/2)$ , there is a polynomial gap between even our monotone lower bounds  $\tilde{\Omega}\left(n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon)}}\right)$ , and upper bounds  $O\left(n^{2 + \frac{\ln(1-\epsilon)}{\ln(\epsilon) - \ln(1-\epsilon)}}\right)$ .

For constant  $\alpha \in (0, 1/2)$  and  $\epsilon = n^{-\alpha}$ , there is a polynomial gap between our lower bounds  $\tilde{\Omega}(n^{2-3\alpha})$  and our upper bounds  $\tilde{O}(n^{2-2\alpha})$ .

4. What is the optimal size for depth greater than 3?

Chaudhuri and Radhakrishnan [6] gave size lower bounds of roughly  $\Omega\left(n^{1 + \frac{1}{2^{2d}}}\right)$  for depth- $d$   $\epsilon$ -promise majority circuits, while we only achieve upper bounds of roughly  $\Omega\left(n^{1 + \frac{1}{2^{d/2-1}}}\right)$ .

5. Do these bounds extend to AC0 with parity, or other circuit classes below TC0?

6. Are there uniform depth-3 circuits for promise majority with the same size as Ajtai's construction? Can we get uniform, depth-4, quadratic size circuits for promise majority?

---

### References

- 1 Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, SFCS '78, page 75–83, USA, 1978. IEEE Computer Society.
- 2 Miklós Ajtai. Sigma11-formulae on finite structures. *Ann. Pure Appl. Log.*, 24:1–48, 1983.
- 3 Miklós Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances In Computational Complexity Theory*, volume 13, pages 1–20, 1993.
- 4 Miklós Ajtai and Michael Ben-Or. A theorem on probabilistic constant depth computations. In *STOC '84*, pages 471–474, 1984.

- 5 Kazuyuki Amano. Bounds on the size of small depth circuits for approximating majority. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, ICALP '09*, page 59–70, Berlin, Heidelberg, 2009. Springer-Verlag.
- 6 Shiva Chaudhuri and Jaikumar Radhakrishnan. Deterministic restrictions in circuit complexity. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96*, page 30–36, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237824.
- 7 Joshua Cook. Size bounds on low depth circuits for promise majority. In *The Electronic Colloquium on Computational Complexity*, 2020.
- 8 Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. In *To appear in The proceedings of the 61st IEEE Symposium on Foundations of Computer Science*, 2020.
- 9 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, STOC '86*, page 6–20, New York, NY, USA, 1986. Association for Computing Machinery.
- 10 Johan Håstad, Ingo Wegener, Norbert Wurm, and Sang-Zin. Yi. Optimal depth, very small size circuits for symmetrical functions in ac0. *Information and Computation*, 108(2):200–211, 1994.
- 11 Nutan Limaye, Srikanth Srinivasan, and Utkarsh Tripathi. More on  $AC^0[\oplus]$  and variants of the majority function. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150, pages 22:1–22:14, 2019.
- 12 Ryan O'Donnell and Karl Wimmer. Approximation by dnf: Examples and counterexamples. In *Proceedings of the 34th International Conference on Automata, Languages and Programming, ICALP'07*, page 195–206, Berlin, Heidelberg, 2007. Springer-Verlag.
- 13 Prabhakar Ragde and Avi Wigderson. Linear-size constant-depth polylog-threshold circuits. *Information Processing Letters*, 39:143–146, 1991.
- 14 Emanuele Viola. On approximate majority and probabilistic time. *Computational Complexity*, 18:337–375, 2009.
- 15 Emanuele Viola. Randomness buys depth for approximate counting. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 230–239, 2011.

# Lower Bounds for Semi-adaptive Data Structures via Corruption

**Pavel Dvořák**

Charles University, Prague, Czech Republic  
koblich@iuuk.mff.cuni.cz

**Bruno Loff**

INESC-Tec and University of Porto, Portugal  
bruno.loff@gmail.com

---

## Abstract

In a dynamic data structure problem we wish to maintain an encoding of some data in memory, in such a way that we may efficiently carry out a sequence of queries and updates to the data. A long-standing open problem in this area is to prove an unconditional polynomial lower bound of a trade-off between the update time and the query time of an adaptive dynamic data structure computing some explicit function. Ko and Weinstein provided such lower bound for a restricted class of *semi-adaptive* data structures, which compute the Disjointness function. There, the data are subsets  $x_1, \dots, x_k$  and  $y$  of  $\{1, \dots, n\}$ , the updates can modify  $y$  (by inserting and removing elements), and the queries are an index  $i \in \{1, \dots, k\}$  (query  $i$  should answer whether  $x_i$  and  $y$  are disjoint, i.e., it should compute the Disjointness function applied to  $(x_i, y)$ ). The semi-adaptiveness places a restriction in how the data structure can be accessed in order to answer a query. We generalize the lower bound of Ko and Weinstein to work not just for the Disjointness, but for any function having high complexity under the smooth corruption bound.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational complexity and cryptography

**Keywords and phrases** semi-adaptive dynamic data structure, polynomial lower bound, corruption bound, information theory

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.20

**Funding** *Pavel Dvořák*: Supported by Czech Science Foundation GAČR (grant #19-27871X).

*Bruno Loff*: This project was financed by the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, through national funds, and co-funded by the FEDER, where applicable. Bruno Loff was the recipient of the postdoctoral FCT fellowship number SFRH/BPD/116010/2016.

**Acknowledgements** We would like to thank Michal Koucký, who worked with us on this paper until the coronavirus pandemic forced him busily away. We would also like to thank Arkadev Chattopadhyay for helpful pointers.

## 1 Introduction

In a dynamic data structure problem we wish to maintain an encoding of some data in memory, in such a way that we may efficiently carry out a sequence of queries and updates to the data. A suitable computational model to study dynamic data structures is the cell probe model of Yao [21]. Here we think of the memory divided into registers, or *cells*, where each cell can carry  $w$  bits, and we measure efficiency by counting the number of memory accesses, or *probes*, needed for each query and each update – these are respectively called the *query time*  $t_q$  and *update time*  $t_u$ . The main goal of this line of research is to understand the inherent trade-off between  $w$ ,  $t_q$  and  $t_u$ , for various interesting problems. Specifically, one would like to show lower bounds on  $t = \max\{t_q, t_u\}$  for reasonable choices of  $w$  (which is typically logarithmic in the size of the data).



© Pavel Dvořák and Bruno Loff;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 20; pp. 20:1–20:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The first lower bound for this setting was proven by Fredman and Saks [8], which proved  $t = \Omega(\log n / \log \log n)$  for various problems. These lower bounds were successively improved [15, 17, 13, 14], and we are now able to show that certain problems with non-Boolean queries require  $t = \Omega((\log n / \log \log n)^2)$ , and certain problems with Boolean queries require  $t = \Omega((\log n / \log \log n)^{3/2})$ .

The major unsolved question in this area is to prove a polynomial lower bound on  $t$ . For example, consider the dynamic reachability problem, where we wish to maintain a directed  $n$ -vertex graph in memory, under edge insertions and deletions, while being able to answer reachability queries (“*is vertex  $i$  connected to vertex  $j$ ?*”). Is it true that any scheme for the dynamic reachability problem requires  $t = \Omega(n^\delta)$ , for some constant  $\delta > 0$ ? Indeed, such a lower bound is known under various complexity-theoretic assumptions<sup>1</sup>, the question is whether such a lower bound may be proven unconditionally.

In an influential paper [18], Mihai Pătraşcu proposed an approach to this unsolved question. He defines a data structure problem, called the *multiphase problem*. Let us represent partial functions  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  as total functions  $f' : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  where  $f'(x, y) = *$  if  $f(x, y)$  is not defined. Then associated with a partial Boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$ , and a natural number  $k \geq 1$ , we may define a corresponding *multiphase problem of  $f$*  as the following dynamic process:

**Phase I - Initialization.** We are given  $k$  inputs  $x_1, \dots, x_k \in \{0, 1\}^n$ , and are allowed to preprocess this input in time  $nk \cdot t_p$ .

**Phase II - Update.** We are then given another input  $y \in \{0, 1\}^n$ , and we have time  $n \cdot t_u$  to read and update the memory locations from the data structure constructed in Phase I.

**Phase III - Query.** Finally, we are given a query  $i \in [k]$ , we have time  $t_q$  to answer the question whether  $f(x_i, y) = 1$ . If  $f(x_i, y)$  is not defined, the answer can be arbitrary.

Typically we will have  $k = \text{poly}(n)$ . Let us be more precise, and consider randomized solutions to the above problem.

► **Definition 1** (Scheme for the multiphase problem of  $f$ ). *Let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  be a partial Boolean function. A scheme for the multiphase problem of  $f$  with preprocessing time  $t_p$ , update time  $t_u$  and query time  $t_q$  is a triple  $D = (E, \{U_y\}_{y \in \{0, 1\}^n}, \{Q_i\}_{i \in [k]})$ , where:*

- $E : (\{0, 1\}^n)^k \rightarrow (\{0, 1\}^w)^s$  maps the input  $x$  to the memory contents  $E(x)$ , where each of the  $s$  memory locations holds  $w$  bits.  $E$  must be computed in time  $nk \cdot t_p$  by a Random-Access Machine (RAM).
- For each  $y \in \{0, 1\}^n$ ,  $U_y : (\{0, 1\}^w)^s \rightarrow (\{0, 1\}^w)^u$  is a decision-tree of depth  $\leq n \cdot t_u$ , which reads  $E(x)$  and produces a sequence  $U_y(E(x))$  of  $u$  updates.<sup>2</sup>
- For each  $i \in [k]$ ,  $Q_i : (\{0, 1\}^w)^s \times (\{0, 1\}^w)^u \rightarrow \{0, 1\}$  is a decision-tree of depth  $\leq t_q$ .<sup>3</sup>
- For all  $x \in (\{0, 1\}^n)^k$ ,  $y \in \{0, 1\}^n$ , and  $i \in [k]$ ,

$$f(x_i, y) \neq * \implies Q_i(E(x), U_y(E(x))) = f(x_i, y).$$

<sup>1</sup> See [16, 1]. Strictly speaking, these conditional lower bounds only work if the preprocessing time, which is the time taken to encode the data into memory, is also bounded. But we will ignore this distinction.

<sup>2</sup> In the usual way of defining the update phase, we have a read/write decision-tree  $U_y$  which changes the very same cells that it reads. But when  $w = \Omega(\log s)$ , this can be seen to be equivalent, up to constant factors, to the definition we present here, where we have a decision-tree  $U_y$  that writes the updates on a separate location. In order to simulate a scheme that uses a read/write decision-tree, we may use a hash table with  $O(1)$  worst-case lookup time, such as cuckoo hashing. Then we have a read-only decision-tree  $U'_y(E(x))$  whose output is the hash table containing all the  $i \in [s]$  which were updated by  $U_y(E(x))$ , associated with their final value in the execution of  $U_y(E(x))$ . Note that the hash table itself is static.

<sup>3</sup> All our results will hold even if  $Q_i$  is allowed to depend arbitrarily on  $x_i$ . This makes for a less natural model, however, so we omit this from the definitions.

In a randomized scheme for the multiphase problem of  $f$ , each  $U_y$  and  $Q_i$  are distributions over decision trees, and it must hold that for all  $x \in (\{0, 1\}^n)^k$ ,  $y \in \{0, 1\}^n$ , and  $i \in [k]$ ,

$$f(x_i, y) \neq * \implies \Pr_{Q_i, U_y} [Q_i(E(x), U_y(E(x))) = f(x_i, y)] \geq 1 - \varepsilon.$$

The value  $\varepsilon$  is called the error probability of the scheme.

Pătraşcu [18] considered this problem where  $f = \text{DISJ}$  is the Disjointness function:

$$\text{DISJ}(x, y) = \begin{cases} 0 & \text{if there exists } i \in [n] \text{ such that } x_i = y_i = 1 \\ 1 & \text{otherwise} \end{cases}$$

He conjectured that any scheme for the multiphase problem of DISJ will necessarily have  $\max\{t_p, t_u, t_q\} \geq n^\delta$  for some constant  $\delta > 0$ .

Pătraşcu shows that such lower bounds on the multiphase problem for DISJ would imply polynomial lower bounds for various dynamic data structure problems. For example such lower bounds would imply that dynamic reachability requires  $t = \Omega(n^\delta)$ . He also shows that these lower bounds hold true under the assumption that 3SUM has no sub-quadratic algorithms.

Finally, Pătraşcu then defines a 3-player Number-On-Forehead (NOF) communication game, such that lower bounds on this game imply matching lower bounds for the multiphase problem. The game associated with a function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  is as follows:

1. Alice is given  $x_1, \dots, x_k \in \{0, 1\}^n$  and  $i \in [k]$ , Bob gets  $y \in \{0, 1\}^n$  and  $i \in [k]$  and Charlie gets  $x_1, \dots, x_k$  and  $y$ .
2. Charlie sends a private message of  $\ell_1$  bits to Bob and then he is silent.
3. Alice and Bob communicate  $\ell_2$  bits and want to compute  $f(x_i, y)$ .

Pătraşcu [18] conjectured that if  $\ell_1 = o(k)$ , then  $\ell_2$  has to be bigger than the communication complexity of  $f$ . However, this conjecture turned out to be false. The randomized communication complexity of DISJ is  $\Omega(n)$  [19, 10, 3], but Chattopadhyay et al. [6] construct a protocol for  $f = \text{DISJ}$  where both  $\ell_1, \ell_2 = O(\sqrt{n} \cdot \log k)$ . They further show that any randomized scheme in the above model can be derandomized.

So the above communication model is more powerful than it appears at first glance.<sup>4</sup> However, a recent paper by Ko and Weinstein [11] succeeds in proving lower bounds for a simpler version of the multiphase problem, which translate to lower bounds for a restricted class of dynamic data structure schemes. They manage to prove a lower bound of  $\Omega(\sqrt{n})$  for the simpler version of the multiphase problem which is associated with the Disjointness function  $f = \text{DISJ}$ . Our paper generalizes their result:

- We generalize their lower bound to any function  $f$  having large complexity according to the smooth corruption bound, under a product distribution. Disjointness is such a function [2], but so is Inner Product, Gap Orthogonality, and Gap Hamming Distance [20].
- The new lower-bounds we obtain (for Inner-product, Gap Orthogonality, and Gap Hamming Distance) are stronger –  $\Omega(n)$  instead of the lower-bound  $\Omega(\sqrt{n})$  for disjointness. As far as was known before our result, it could well have been that every function had a scheme for the simpler version of the multiphase problem using only  $O(\sqrt{n})$  communication.

<sup>4</sup> The conjecture remains that if  $\ell_1 = o(k)$ , then  $\ell_2$  has to be larger than the maximum distributional communication complexity of  $f$  under a product distribution. This is  $\tilde{\Theta}(\sqrt{n})$  for Disjointness [2].

- Ko and Weinstein derive their lower-bound via a cut-and-paste lemma which works specifically for disjointness. This cut-and-paste lemma is a more robust version of the one appearing in [3], made to work not only for protocols, where the inputs  $\mathbf{x}$  and  $\mathbf{y}$  are independent given the transcript  $\mathbf{z}$  of the protocol, but also for random-variables that are “protocol-like”, namely any  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  where  $I(\mathbf{x} : \mathbf{y} \mid \mathbf{z})$  is close to 0. Instead, we directly derive the existence of a large nearly-monochromatic rectangle, from the existence of such protocol-like random-variables, which is what then allows us to use the smooth corruption bound. This result is our core technical contribution, and may be of independent interest.

All of the above lower bounds will be shown to hold also for randomized schemes, and not just for deterministic schemes.

## 1.1 Semi-adaptive Multiphase Problem

Let us provide rigorous definitions.

► **Definition 2** (Semi-adaptive random data structure [11]). *Let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  be a partial function. A scheme  $D = (E, \{U_y\}_{y \in \{0, 1\}^n}, \{Q_i\}_{i \in [k]})$  for the multiphase problem of  $f$  is called semi-adaptive if any path on the decision-tree  $Q_i : (\{0, 1\}^w)^s \times (\{0, 1\}^w)^u \rightarrow \{0, 1\}$  first queries the first part of the input (the  $E(x)$  part), and then queries the second part of the input (the  $U(E(x))$  part). If  $D$  is randomized, then this property must hold for every randomized choice of  $Q_i$ .*

We point out that the reading of the cells in each part is completely adaptive. The restriction is only that the data structure cannot read cells of  $E(x)$  if it already started to read cells of  $U(E(x))$ . Ko and Weinstein state their result for deterministic data structures, i.e.,  $\varepsilon = 0$  thus the data structure always returns the correct answer.

► **Theorem 3** (Theorem 4.9 of Ko and Weinstein [11]). *Let  $k \geq \omega(n)$ . Any semi-adaptive deterministic data structure that solves the multiphase problem of the DISJ function, must have either  $t_u \cdot n \geq \Omega(k/w)$  or  $t_q \geq \Omega(\sqrt{n}/w)$ .*

To prove the lower bound they reduce the semi-adaptive data structure into a low correlation random process.

► **Theorem 4** (Reformulation of Lemma 4.1 of Ko and Weinstein [11]). *Let  $\mathbf{x}_1, \dots, \mathbf{x}_k$  be random variables over  $\{0, 1\}^n$  and each of them is independently distributed according to the same distribution  $\mu_1$  and let  $\mathbf{y}$  be a random variable over  $\{0, 1\}^n$  distributed according to  $\mu_2$  (independently of  $\mathbf{x}_1, \dots, \mathbf{x}_k$ ). Let  $D$  be a randomized semi-adaptive scheme for the multiphase problem for a partial function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  with error probability bounded by  $\varepsilon$ . Then, for any  $p \leq o(k)$  there is a random variable  $\mathbf{z} \in \{0, 1\}^m$  and  $i \in [k]$  such that:*

1.  $\Pr[f(\mathbf{x}_i, \mathbf{y}) \neq *, \mathbf{z}_m \neq f(\mathbf{x}_i, \mathbf{y})] \leq \varepsilon$ .
2.  $I(\mathbf{x}_i : \mathbf{y} \mid \mathbf{z}) \leq t_q \cdot w + o(t_q \cdot w)$ .
3.  $I(\mathbf{y} : \mathbf{z}) \leq t_q \cdot w$ .
4.  $I(\mathbf{x}_i : \mathbf{y} \mid \mathbf{z}) \leq O\left(\frac{t_u \cdot n \cdot w}{p}\right)$ .

Above,  $I$  stands for mutual information between random variables, see Section 2.2 for the definition. The random variable  $\mathbf{z}$  consists of some  $\mathbf{x}_j$ 's and transcripts of query phases of  $D$  for some  $j \in [k]$ . The theorem can be interpreted as saying that the last bit of  $\mathbf{z}$  predicts  $f(\mathbf{x}_i, \mathbf{y})$ ,  $\mathbf{z}$  has little information about  $\mathbf{x}_i$  and  $\mathbf{y}$ , and the triple  $(\mathbf{x}_i, \mathbf{y}, \mathbf{z})$  is “protocol-like”, in the sense that  $\mathbf{x}_i$  and  $\mathbf{y}$  are close to being independent given  $\mathbf{z}$ . Ko and Weinstein [11]

proved Theorem 4 for the deterministic schemes for the DISJ function and in the case where  $\mu_1 = \mu_2$ . However, their proof actually works for any (partial) function  $f$  and for any two, possibly distinct distributions  $\mu_1$  and  $\mu_2$ . Moreover, their proof also works for randomized schemes. The resulting statement for randomized schemes for any function  $f$  is what we have given above. To complete the proof of their lower bound, Ko and Weinstein proved that if we set  $p$  (and  $k$ ) large enough so that  $I(\mathbf{x}_i : \mathbf{y} \mid \mathbf{z}) \leq o(1)$  then such random variable  $\mathbf{z}$  cannot exist when  $f$  is the DISJ function. It is this second step which we generalize.

Let  $f : X \times Y \rightarrow \{0, 1\}$  be a function and  $\mu$  be a distribution over  $X \times Y$ . A set  $R \subseteq X \times Y$  is a *rectangle* if there exist sets  $A \subseteq X$  and  $B \subseteq Y$  such that  $R = A \times B$ . For  $b \in \{0, 1\}$  and  $0 \leq \rho \leq 1$ , we say the rectangle  $R$  is  $\rho$ -error  $b$ -monochromatic for  $f$  under  $\mu$  if  $\mu(R \cap f^{-1}(1-b)) \leq \rho \cdot \mu(R)$ . We say the distribution  $\mu$  is a *product distribution* if there are two independent distribution  $\mu_1$  over  $X$  and  $\mu_2$  over  $Y$  such that  $\mu(x, y) = \mu_1(x) \times \mu_2(y)$ . For  $0 \leq \alpha \leq \frac{1}{2}$ , the distribution  $\mu$  is  $\alpha$ -balanced according to  $f$  if  $\mu(f^{-1}(0)), \mu(f^{-1}(1)) \geq \alpha$ . We will prove that the existence of a random variable  $\mathbf{z}$  given by Theorem 4 implies that, for any  $b \in \{0, 1\}$ , any balanced product distribution  $\mu$  and any function  $g$  which is “close” to  $f$ , there is a large (according to  $\mu$ )  $\rho$ -error  $b$ -monochromatic rectangle for  $g$  in terms of  $t_q$ . This technique is known as smooth corruption bound [4, 5] or smooth rectangle bound [9]. We denote the smooth corruption bound of  $f$  as  $\text{scb}_\mu^{\rho, \lambda}$ . Informally,  $\text{scb}_\mu^{\rho, \lambda}(f) \geq s$  if there is  $b \in \{0, 1\}$  and a partial function  $g : X \times Y \rightarrow \{0, 1, *\}$  which is close<sup>5</sup> to  $f$  such that any  $\rho$ -error  $b$ -monochromatic rectangle  $R \subseteq X \times Y$  for  $g$  has size (under  $\mu$ ) at most  $2^{-s}$ . We will define smooth corruption bound formally in the next section. Thus, if we use Theorem 4 as a black box we generalize Theorem 3 for any function of large corruption bound.

► **Theorem 5 (Main Result).** *Let  $\lambda, \tilde{\varepsilon}, \tilde{\alpha} \geq 0$  such that  $\alpha \geq 2\varepsilon$  for  $\varepsilon = \tilde{\varepsilon} + \lambda, \alpha = \tilde{\alpha} - \lambda$ . Let  $\mu$  be a product distribution over  $\{0, 1\}^n \times \{0, 1\}^n$  such that  $\mu$  is  $\tilde{\alpha}$ -balanced according to a partial function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$ . Any semi-adaptive randomized scheme for the multiphase problem of  $f$ , with error probability bounded by  $\tilde{\varepsilon}$ , must have either  $t_u \cdot n \geq \Omega(k/w)$ , or*

$$t_q \cdot w \geq \Omega\left(\alpha \cdot \text{scb}_\mu^{O(\varepsilon/\alpha), \lambda}(f)\right).$$

We point out that  $\Omega$  and  $O$  in the bound given above hide absolute constants independent of  $\alpha, \varepsilon$  and  $\lambda$ .

As a consequence of our main result, and of previously-known bounds on corruption, we are able to show new lower-bounds of  $t_q = \Omega(\frac{n}{w})$  against semi-adaptive schemes for the multiphase problem of the Inner Product, Gap Orthogonality and Gap Hamming Distance functions (where the gap is  $\sqrt{n}$ ). These lower-bounds hold assuming that  $t_u = o(\frac{k}{wn})$ . They follow from the small discrepancy of Inner Product, and from a bound shown by Sherstov on the corruption of Gap Orthogonality, followed by a reduction to Gap Hamming Distance [20]. This result also gives an alternative proof of the same lower-bound proven by Ko and Weinstein [11], for the Disjointness function, of  $t_q = \Omega(\frac{\sqrt{n}}{w})$ . This follows from the bound on corruption of Disjointness under a product distribution, shown by Babai et al. [2].

The paper is organized as follows. In Section 2 we give important notation, and the basic definitions from information theory and communication complexity. The proof of Theorem 5 appears in Section 3. The various applications appear in Section 4.

<sup>5</sup> “Closeness” is measured by the parameter  $\lambda \in \mathbb{R}$ , see Section 2.1 for the formal definition.

## 2 Preliminaries

We use a notational scheme where sets are denoted by uppercase letters, such as  $X$  and  $Y$ , elements of the sets are denoted by the same lowercase letters, such as  $x \in X$  and  $y \in Y$ , and random variables are denoted by the same lowercase boldface letters, such as  $\mathbf{x}$  and  $\mathbf{y}$ . We will use lowercase greek letters, such as  $\mu$ , to denote distributions. If  $\mu$  is a distribution over a product set, such as  $X \times Y \times Z$ , and  $(x, y, z) \in X \times Y \times Z$ , then  $\mu(x, y, z)$  is the probability of seeing  $(x, y, z)$  under  $\mu$ . We will sometimes denote  $\mu$  by  $\mu(\mathbf{x}, \mathbf{y}, \mathbf{z})$ , using non-italicized lowercase letters corresponding to  $X \times Y \times Z$ . This allows us to use the notation  $\mu(\mathbf{x})$  and  $\mu(\mathbf{y})$  to denote the  $x$  and  $y$ -marginals of  $\mu$ , for example; then if we use the same notation with italicized lowercase letters, we get the marginal probabilities, i.e., for each  $x \in X$  and  $y \in Y$

$$\mu(x) = \sum_{y,z} \mu(x, y, z) \quad \mu(y) = \sum_{x,z} \mu(x, y, z).$$

If  $y \in Y$ , then we will also use the notation  $\mu(x | y)$  to denote the  $x$ -marginal of  $\mu$  conditioned seeing the specific value  $y$ . Then for each  $x \in X$  and  $y \in Y$ , we have

$$\mu(x | y) = \sum_z \mu(x, y, z).$$

We will also write  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \sim \mu$  to mean that  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  are random variables chosen according to the distribution  $\mu(\mathbf{x}, \mathbf{y}, \mathbf{z})$ , i.e., for all  $(x, y, z) \in X \times Y \times Z$ ,  $\Pr[\mathbf{x} = x, \mathbf{y} = y, \mathbf{z} = z] = \mu(x, y, z)$ . Naturally if  $A \subseteq X \times Y \times Z$ , then  $\mu(A) = \sum_{(x,y,z) \in A} \mu(x, y, z)$ . We let  $\text{supp}(\mu)$  denote the *support* of  $\mu$ , i.e., the set of  $(x, y, z)$  with  $\mu(x, y, z) > 0$ .

We now formally define the smooth corruption bound and related measures from communication complexity, and refer the book by Kushilevitz and Nisan [12] for more details. At the end of this section we provide necessary notions of information theory which are used in the paper, and for more details on these we refer to the book by Cover and Thomas [7].

### 2.1 Rectangle Measures

Let  $f : X \times Y \rightarrow \{0, 1, *\}$  be a partial function, where  $f(x, y) = *$  means  $f$  is not defined on  $(x, y)$ . Let  $\mu(\mathbf{x}, \mathbf{y})$  be a distribution over  $X \times Y$ . We say that  $f$  is  $\lambda$ -close to a partial function  $g : X \times Y \rightarrow \{0, 1, *\}$  under  $\mu$  if

$$\Pr_{(x,y) \sim \mu} [f(x, y) \neq g(x, y)] \leq \lambda.$$

For  $b \in \{0, 1\}$ ,  $\rho \in [0, 1]$ , let

$$\mathcal{R}_\mu^{\rho, b}(f) = \{R \subseteq X \times Y \text{ rectangle} \mid \mu(R \cap f^{-1}(1-b)) \leq \rho \cdot \mu(R)\}$$

be the set of  $\rho$ -error  $b$ -monochromatic rectangles for  $f$  under  $\mu$ . The complexity measure  $\text{mono}$  quantifies how large almost  $b$ -monochromatic rectangles can be for both  $b \in \{0, 1\}$ :

$$\text{mono}_\mu^\rho(f) = \min_{b \in \{0, 1\}} \max_{R \in \mathcal{R}_\mu^{\rho, b}(f)} \mu(R)$$

Using  $\text{mono}$  we can define the *corruption bound* of a function as  $\text{cb}_\mu^\rho(f) = \log \frac{1}{\text{mono}_\mu^\rho(f)}$  and the *smooth corruption bound* as

$$\text{scb}_\mu^{\rho, \lambda}(f) = \max_{g: \lambda\text{-close to } f \text{ under } \mu} \text{cb}_\mu^\rho(g).$$

Thus, if  $\text{scb}_\mu^{\rho, \lambda}(f) \geq s$  then there is a  $b \in \{0, 1\}$  and a function  $g$  which  $\lambda$ -close to  $f$  under  $\mu$  such that for any  $\rho$ -error  $b$ -monochromatic rectangle for  $g$  under  $\mu$  it holds that  $\mu(R) \leq 2^{-s}$ .

► **Remark.** In Razborov’s paper where an  $\Omega(n)$  lower-bound for disjointness is first proven [19], the (implicitly given) definition of a  $\rho$ -error  $b$ -monochromatic rectangle is  $\mu(R \cap f^{-1}(1 - b)) \leq \rho \cdot \mu(R \cap f^{-1}(b))$ . Later, a strong direct product theorem for corruption (under product distributions) was proven by Beame et al. [4], which uses instead the condition that  $\mu(R \setminus f^{-1}(b)) \leq \rho \cdot \mu(R)$ . The definition we present above comes from [20], where the condition is (we repeat it here) that  $\mu(R \cap f^{-1}(1 - b)) \leq \rho \cdot \mu(R)$ . So we have three different definitions of  $\rho$ -error  $b$ -monochromatic rectangle, and thus three different corruption bounds. Now, if the distribution  $\mu$  is supported on the domain of  $f$ , all these three definitions result in (roughly) equivalent complexity measures. But if  $\mu$  attributes some mass to inputs where  $f$  is undefined (which is sometimes useful if  $\mu$  is a product distribution, as in our case), then the definitions are no longer equivalent. Our lower-bound will hold for any of the definitions, but the proof is somewhat simpler for the definition used in Sherstov’s paper [20], which is the only corruption-based lower-bound we use, where  $\mu$  attributes mass to undefined inputs.

The notion  $\text{mono}_\mu^\rho$  is related to the *discrepancy* of a function:

$$\text{disc}_\mu(f) = \max_{R: \text{rectangle of } X \times Y} \left| \mu(R \cap f^{-1}(0)) - \mu(R \cap f^{-1}(1)) \right|.$$

It is easy to see that for a total function  $f$  holds that  $\text{disc}_\mu(f) \geq (1 - 2\rho) \cdot \text{mono}_\mu^\rho(f)$  for any  $\rho$ . Thus, Theorem 5 will give us lower bounds also for functions of small discrepancy.

## 2.2 Information Theory

We define several measures from information theory. If  $\mu'(z), \mu(z)$  are two distributions such that  $\text{supp}(\mu') \subseteq \text{supp}(\mu)$ , then the *Kullback-Leibler divergence* of  $\mu'$  from  $\mu$  is

$$D_{\text{KL}}(\mu' \parallel \mu) = \sum_z \mu'(z) \log \frac{\mu'(z)}{\mu(z)}.$$

With Kullback-Leibler divergence we can define the mutual information, which measures how close (according to KL divergence) is a joint distribution to the product of its marginals. If we have two random variables  $(\mathbf{x}, \mathbf{y}) \sim \mu(\mathbf{x}, \mathbf{y})$ , then we define their *mutual information* to be

$$I(\mathbf{x} : \mathbf{y}) = D_{\text{KL}}(\mu(\mathbf{x}, \mathbf{y}) \parallel \mu(\mathbf{x}) \times \mu(\mathbf{y})) = \mathbb{E}_{y \sim \mu(y)} \left[ D_{\text{KL}}(\mu(\mathbf{x} \mid y) \parallel \mu(\mathbf{x})) \right].$$

If we have three random variables  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \sim \mu(\mathbf{x}, \mathbf{y}, \mathbf{z})$ , then the *mutual information of  $\mathbf{x}$  and  $\mathbf{y}$  conditioned by  $\mathbf{z}$*  is

$$I(\mathbf{x} : \mathbf{y} \mid \mathbf{z}) = \mathbb{E}_{z \sim \mu(z)} \left[ I(\mathbf{x} : \mathbf{y} \mid \mathbf{z} = z) \right] = \mathbb{E}_{z \sim \mu(z)} \left[ D_{\text{KL}}(\mu(\mathbf{x}, \mathbf{y} \mid z) \parallel \mu(\mathbf{x} \mid z) \times \mu(\mathbf{y} \mid z)) \right]$$

We present several facts about mutual information, the proofs can be found in the book of Cover and Thomas [7].

► **Fact 6 (Chain Rule).** For any random variables  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}$  and  $\mathbf{z}$  holds that

$$I(\mathbf{x}_1 \mathbf{x}_2 : \mathbf{y} \mid \mathbf{z}) = I(\mathbf{x}_1 : \mathbf{y} \mid \mathbf{z}) + I(\mathbf{x}_2 : \mathbf{y} \mid \mathbf{z}, \mathbf{x}_1).$$

Since mutual information is never negative, we have the following corollary.

► **Corollary 7.** For any random variables  $\mathbf{x}, \mathbf{y}$  and  $\mathbf{z}$  holds that  $I(\mathbf{x} : \mathbf{y}) \leq I(\mathbf{x} : \mathbf{y} \mathbf{z})$ .

The  $\ell_1$ -distance between two distributions is defined as

$$\|\mu'(z) - \mu(z)\|_1 = \sum_z |\mu'(z) - \mu(z)|.$$

There is a relation between  $\ell_1$ -distance and Kullback-Leibler divergence.

► **Fact 8** (Pinsker's Inequality). *For any two distributions  $\mu'(z)$  and  $\mu(z)$ , we have*

$$\|\mu'(z) - \mu(z)\|_1 \leq \sqrt{2 \cdot D_{\text{KL}}(\mu'(z) \parallel \mu(z))}$$

### 3 The Proof of Theorem 5

Let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  be a partial function. Suppose there is a semi-adaptive random scheme  $D$  for the multiphase problem of  $f$  with error probability bounded by  $\tilde{\varepsilon}$  such that  $t_u \cdot n \leq o(k/w)$ . Let  $\mu(x, y) = \mu_1(x) \times \mu_2(y)$  be a product distribution over  $\{0, 1\}^n \times \{0, 1\}^n$ , such that  $\mu(x, y)$  is  $\tilde{\alpha}$ -balanced according to  $f$ . Let  $b \in \{0, 1\}$  and  $g : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, *\}$  be a partial function which is  $\lambda$ -close to  $f$  under  $\mu$ . We will prove there is a large almost  $b$ -monochromatic rectangle for  $g$ .

Let  $\mathbf{x}_1, \dots, \mathbf{x}_k$  be independent random variables each of them distributed according to  $\mu_1$  and  $\mathbf{y}$  be an independent random variable distributed according to  $\mu_2$ . Let the random variable  $\mathbf{z} \in \{0, 1\}^m$  and the index  $i \in [k]$  be given by Theorem 4 applied to the random variables  $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y}$  and the function  $f$ . For simplicity we denote  $\mathbf{x} = \mathbf{x}_i$ .

We will denote the joint distribution of  $(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y}, \mathbf{z})$  by  $\mu(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y}, \mathbf{z})$ . Note that here the notation is consistent, in the sense that  $\mu(x_i, y) = \mu_1(x_i) \times \mu_2(y)$  for all  $i \in [k], x, y \in \{0, 1\}^n$ . We will then need to keep in mind that  $\mu(z)$  is the  $z$ -marginal of the joint distribution of  $(\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y}, \mathbf{z})$ .

By  $f(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m$  we denote the event that the random variable  $\mathbf{z}_m$  gives us the wrong answer on an input from the support of  $f$ , i.e.  $f(\mathbf{x}, \mathbf{y}) \neq *$  and  $f(\mathbf{x}, \mathbf{y}) \neq \mathbf{z}_m$  hold simultaneously. By Theorem 4 we know that  $\Pr[f(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m] \leq \tilde{\varepsilon}$ . Since  $f$  and  $g$  are  $\lambda$ -close under  $\mu$ , we have that  $\mu$  is still balanced according to  $g$  and  $g(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m$  with small probability, as stated in the next observation.

► **Observation 9.** *Let  $\alpha = \tilde{\alpha} - \lambda$  and  $\varepsilon = \tilde{\varepsilon} + \lambda$ . For the function  $g$  it holds that*

1. *The distribution  $\mu(x, y)$  is  $\alpha$ -balanced according to  $g$ .*
2.  $\Pr[g(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m] \leq \varepsilon$ .

**Proof.** Let  $b' \in \{0, 1\}$ . We will bound  $\mu(g^{-1}(b'))$ .

$$\begin{aligned} \tilde{\alpha} &\leq \Pr[f(\mathbf{x}, \mathbf{y}) = b'] = \Pr[f(\mathbf{x}, \mathbf{y}) = b', f(\mathbf{x}, \mathbf{y}) = g(\mathbf{x}, \mathbf{y})] \\ &\quad + \Pr[f(\mathbf{x}, \mathbf{y}) = b', f(\mathbf{x}, \mathbf{y}) \neq g(\mathbf{x}, \mathbf{y})] \\ &\leq \Pr[g(\mathbf{x}, \mathbf{y}) = b'] + \lambda. \end{aligned}$$

Thus, by rearranging we get  $\mu(g^{-1}(b')) \geq \tilde{\alpha} - \lambda = \alpha$ . The proof of the second bound is similar:

$$\begin{aligned} \Pr[g(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m] &= \Pr[f(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m, f(\mathbf{x}, \mathbf{y}) = g(\mathbf{x}, \mathbf{y})] \\ &\quad + \Pr[g(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m, f(\mathbf{x}, \mathbf{y}) \neq g(\mathbf{x}, \mathbf{y})] \leq \tilde{\varepsilon} + \lambda = \varepsilon. \end{aligned} \quad \blacktriangleleft$$

Let  $c$  be the bound on  $I(\mathbf{x} : \mathbf{y} \mathbf{z})$  and  $I(\mathbf{y} : \mathbf{z})$  given by Theorem 4. Since  $I(\mathbf{x} : \mathbf{z}) \leq I(\mathbf{x} : \mathbf{y} \mathbf{z})$ , we have  $I(\mathbf{x} : \mathbf{z}), I(\mathbf{y} : \mathbf{z}) \leq t_q \cdot w + o(t_q \cdot w) = c$ . We will prove that if we assume that  $t_u \cdot n < o(k/w)$  and we choose  $p$  large enough ( $p$  of Theorem 4) then we can find a

rectangle  $R \subseteq X \times Y$  such that  $R$  is  $O(\varepsilon/\alpha)$ -error  $b$ -monochromatic for  $g$  and  $\mu(R) \geq \frac{1}{2^{c'}}$  for  $c' = O\left(\frac{t_q \cdot w}{\alpha}\right)$ . Thus, we have  $\text{mono}_\mu^{O(\varepsilon/\alpha)}(g) \geq 2^{-c'}$  and consequently

$$\text{scb}_\mu^{O(\varepsilon/\alpha), \lambda}(f) \leq O\left(\frac{t_q \cdot w}{\alpha}\right).$$

By rearranging, we get the bound of Theorem 5.

Let us sketch the proof of how we can find such a rectangle  $R$ . We will first fix the random variable  $\mathbf{z}$  to  $z$  such that  $\mathbf{x}$  and  $\mathbf{y}$  are not very correlated conditioned on  $\mathbf{z} = z$ , i.e., the joint distribution  $\mu(\mathbf{x}, \mathbf{y} \mid z)$  is very similar to the product distribution of the marginals  $\mu(\mathbf{x} \mid z) \times \mu(\mathbf{y} \mid z)$ . Moreover, we will pick  $z$  in such a way the probability of error  $\Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m \mid \mathbf{z} = z]$  is still small. Then, since  $\mu(\mathbf{x}, \mathbf{y} \mid z)$  is close to  $\mu(\mathbf{x} \mid z) \times \mu(\mathbf{y} \mid z)$ , the probability of error under the latter distribution will be small as well, i.e., if  $(\mathbf{x}', \mathbf{y}') \sim \mu(\mathbf{x} \mid z) \times \mu(\mathbf{y} \mid z)$ , then  $\Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m]$  will also be small. Finally, we will find subsets  $A \subseteq \text{supp}(\mu(\mathbf{x} \mid z)), B \subseteq \text{supp}(\mu(\mathbf{y} \mid z))$  of large mass (under the original distributions  $\mu_1$  and  $\mu_2$ ), while keeping the probability of error on the rectangle  $R = A \times B$  sufficiently small.

Let us then proceed to implement this plan. Let  $\beta = \alpha - \varepsilon$ . We will show that  $\beta$  is a lower bound for the probability that  $\mathbf{z}_m$  is equal to  $b$ . Let  $\gamma$  be the bound on  $I(\mathbf{x} : \mathbf{y} \mid \mathbf{z})$  given by Theorem 4, i.e.,  $I(\mathbf{x}_i : \mathbf{y} \mid \mathbf{z}) \leq \gamma = O\left(\frac{t_w \cdot n \cdot w}{p}\right)$ .

► **Lemma 10.** *There exists  $z \in Z$  such that*

1.  $z_m = b$ .
2.  $I(\mathbf{x} : \mathbf{y} \mid \mathbf{z} = z) \leq \frac{5}{\beta} \cdot \gamma$ .
3.  $D_{\text{KL}}(\mu(\mathbf{x} \mid z) \parallel \mu(\mathbf{x})), D_{\text{KL}}(\mu(\mathbf{y} \mid z) \parallel \mu(\mathbf{y})) \leq \frac{5}{\beta} \cdot c$ .
4.  $\Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m \mid \mathbf{z} = z] \leq \frac{5}{\beta} \cdot \varepsilon$ .

**Proof.** Since  $\mu$  is  $\alpha$ -balanced according to  $g$ , we find that

$$\begin{aligned} \alpha &\leq \Pr[g(\mathbf{x}, \mathbf{y}) = b] \\ &= \Pr[g(\mathbf{x}, \mathbf{y}) = b, \mathbf{z}_m = b] + \Pr[g(\mathbf{x}, \mathbf{y}) = b, \mathbf{z}_m \neq b] \leq \Pr[\mathbf{z}_m = b] + \varepsilon. \end{aligned}$$

Thus, by rearranging we get  $\Pr[\mathbf{z}_m = b] \geq \alpha - \varepsilon = \beta$ . By expanding the information  $I(\mathbf{x} : \mathbf{y} \mid \mathbf{z})$  we find

$$\gamma \geq I(\mathbf{x} : \mathbf{y} \mid \mathbf{z}) = \mathbb{E}_{z \sim \mu(\mathbf{z})} [I(\mathbf{x} : \mathbf{y} \mid \mathbf{z} = z)]$$

and by the Markov inequality we get that

$$\Pr_{z \sim \mu(\mathbf{z})} \left[ I(\mathbf{x} : \mathbf{y} \mid \mathbf{z} = z) \geq \frac{5}{\beta} \cdot \gamma \right] \leq \frac{\beta}{5}.$$

Similarly, for the information  $I(\mathbf{x} : \mathbf{z})$ :

$$c \geq I(\mathbf{x} \mathbf{y} : \mathbf{z}) \geq I(\mathbf{x} : \mathbf{z}) = \mathbb{E}_{z \sim \mu(\mathbf{z})} [D_{\text{KL}}(\mu(\mathbf{x} \mid z) \parallel \mu(\mathbf{x}))]$$

and so

$$\Pr_{z \sim \mu(\mathbf{z})} \left[ D_{\text{KL}}(\mu(\mathbf{x} \mid z) \parallel \mu(\mathbf{x})) \geq \frac{5}{\beta} \cdot c \right] \leq \frac{\beta}{5}.$$

## 20:10 Lower Bounds for Semi-adaptive Data Structures via Corruption

The bound for  $I(\mathbf{y} : \mathbf{z})$  is analogous. Let  $e_z = \Pr_{\mu}[g(\mathbf{x}, \mathbf{y}) \neq^* z_m | \mathbf{z} = z]$ . Then,

$$\begin{aligned} \varepsilon &\geq \Pr[g(\mathbf{x}, \mathbf{y}) \neq^* \mathbf{z}_m] = \sum_{z \in Z} \mu(z) \cdot e_z = \mathbb{E}_{z \sim \mu(z)} [e_z] \\ \Pr_{z \sim \mu(z)} \left[ e_z \geq \frac{5}{\beta} \cdot \varepsilon \right] &\leq \frac{\beta}{5}. \end{aligned}$$

Thus, by a union bound we may infer the existence of the sought  $z \in Z$ .  $\blacktriangleleft$

Let us now fix  $z \in Z$  from the previous lemma. Let  $\mu_z(x, y) = \mu(x, y | z)$  be the distribution  $\mu(x, y)$  conditioned on  $\mathbf{z} = z$ , and let  $\mu'_z(x, y) = \mu(x | z) \times \mu(y | z)$  be the product of its marginals. Let  $S$  be the support of  $\mu_z(x, y)$ , and let  $S_x$  and  $S_y$  be the supports of  $\mu'_z(x)$  and  $\mu'_z(y)$ , respectively, i.e.,  $S_x$  and  $S_y$  are the projections of  $S$  into  $X$  and  $Y$ .

Then Pinsker's inequality will give us that  $\mu_z$  and  $\mu'_z$  are very close. Let  $\delta = \sqrt{\frac{10}{\beta} \cdot \gamma}$ .

► **Lemma 11.**  $\left\| \mu_z(x, y) - \mu'_z(x, y) \right\|_1 \leq \delta$

**Proof.** Indeed, by Pinsker's inequality,

$$\left\| \mu_z(x, y) - \mu'_z(x, y) \right\|_1 \leq \sqrt{2 \cdot \text{D}_{\text{KL}}(\mu_z(x, y) \parallel \mu'_z(x, y))}.$$

The right-hand side is  $\sqrt{2 \cdot \text{D}_{\text{KL}}(\mu(x, y | z) \parallel \mu(x | z) \times \mu(y | z))}$ , which by definition of mutual information equals  $\sqrt{2 \cdot I(\mathbf{x} : \mathbf{y} | \mathbf{z} = z)}$ , and by Lemma 10 this is  $\leq \sqrt{\frac{10}{\beta} \cdot \gamma} = \delta$ .  $\blacktriangleleft$

For the sake of reasoning, let  $(\mathbf{x}', \mathbf{y}') \sim \mu'_z(x, y)$  be random variables chosen according to  $\mu'_z$ . Let  $\varepsilon' = \frac{5}{\beta} \cdot \varepsilon + \delta$ . It then follows from Lemma 10 and Lemma 11 that:

► **Lemma 12.**  $\Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m] \leq \varepsilon'$ .

**Proof.** We prove that

$$\left| \Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m | \mathbf{z} = z] - \Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m] \right| \leq \delta.$$

Since  $\Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m | \mathbf{z} = z] \leq \frac{5}{\beta} \cdot \varepsilon$  by Lemma 10, the lemma follows. Let

$$B = \{(x, y) \in S_x \times S_y : g(x, y) \neq z_m, g(x, y) \neq *\}.$$

$$\begin{aligned} &\left| \Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m | \mathbf{z} = z] - \Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m] \right| \\ &= \left| \sum_{(x, y) \in B} \mu_z(x, y) - \mu'_z(x, y) \right| \\ &\leq \sum_{(x, y) \in B} \left| \mu_z(x, y) - \mu'_z(x, y) \right| \leq \delta \quad \text{by the triangle inequality and Lemma 11} \end{aligned}$$

$\blacktriangleleft$

Let  $c' = \frac{5}{\beta} \cdot c$ . We will prove the ratio between  $\mu'_z(\mathbf{x}')$  and  $\mu(\mathbf{x}')$  is larger than  $2^{O(c')}$  with only small probability (when  $\mathbf{x}' \sim \mu'_z(x)$ ). The same holds for  $\mu'_z(\mathbf{y}')$  and  $\mu(\mathbf{y}')$ .

► **Lemma 13.**  $\Pr \left[ \mu'_z(\mathbf{x}') \geq 2^{6c'} \cdot \mu(\mathbf{x}') \right], \Pr \left[ \mu'_z(\mathbf{y}') \geq 2^{6c'} \cdot \mu(\mathbf{y}') \right] \leq \frac{1}{6}$ .

**Proof.** We prove the lemma for  $\mu'_z(\mathbf{x}')$ , the proof for  $\mu'_z(\mathbf{y}')$  is analogous. By Lemma 10 we know that  $D_{\text{KL}}(\mu(x | z) \| \mu(x)) = D_{\text{KL}}(\mu_z(x) \| \mu(x)) = D_{\text{KL}}(\mu'_z(x) \| \mu(x)) \leq c'$ . We expand the Kullback-Leibler divergence:

$$c' \geq D_{\text{KL}}(\mu'_z(x) \| \mu(x)) = \sum_{x \in S_x} \mu'_z(x) \log \frac{\mu'_z(x)}{\mu(x)} = \mathbb{E} \left[ \log \frac{\mu'_z(\mathbf{x}')}{\mu(\mathbf{x}')} \right],$$

and then use the Markov inequality:

$$\Pr \left[ \mu'_z(\mathbf{x}') \geq 2^{6c'} \cdot \mu(\mathbf{x}') \right] = \Pr \left[ \log \frac{\mu'_z(\mathbf{x}')}{\mu(\mathbf{x}')} \geq 6c' \right] \leq \frac{1}{6}. \quad \blacktriangleleft$$

We now split  $S_x$  and  $S_y$  into buckets  $C_\ell^x$  and  $C_\ell^y$  (for  $\ell \geq 1$ ), where the  $\ell$ -th buckets are

$$C_\ell^x = \left\{ x \in S_x \mid \frac{(\ell-1)}{2^{c'}} < \frac{\mu'_z(x)}{\mu(x)} \leq \frac{\ell}{2^{c'}} \right\},$$

$$C_\ell^y = \left\{ y \in S_y \mid \frac{(\ell-1)}{2^{c'}} < \frac{\mu'_z(y)}{\mu(y)} \leq \frac{\ell}{2^{c'}} \right\}.$$

In a bucket  $C_\ell^x$  there are elements of  $S_x$  such that their probability under  $\mu'_z(x)$  is approximately  $\frac{\ell}{2^{c'}}$ -times bigger than their probability under  $\mu(x)$ . By Lemma 13, it holds that with high probability the elements  $x \in S_x, y \in S_y$  are in the buckets  $C_{\ell_1}^x$  and  $C_{\ell_2}^y$  for  $\ell_1, \ell_2 \leq 2^{7c'}$ . Thus, if we find a bucket  $C_{\ell_1}^x$  for  $\ell_1 \leq 2^{7c'}$  which has probability at least  $\frac{1}{2^{6c'}}$  under  $\mu'_z(x)$ , then it has also probability at least  $\frac{1}{2^{6c'}}$  under  $\mu(x)$ . The same holds also for buckets  $C_{\ell_2}^y$ . In the next lemma we will show that there are buckets  $C_{\ell_1}^x$  and  $C_{\ell_2}^y$  of large probability under  $\mu'_z$  such that the probability of error on  $C_{\ell_1}^x \times C_{\ell_2}^y$  is still small.

► **Lemma 14.** *There exist buckets  $C_{\ell_1}^x$  and  $C_{\ell_2}^y$  such that*

1.  $1 < \ell_1, \ell_2 \leq 2^{7c'}$ .
2.  $\Pr[\mathbf{x}' \in C_{\ell_1}^x], \Pr[\mathbf{y}' \in C_{\ell_2}^y] \geq \frac{1}{6 \cdot 2^{7c'}}$ .
3.  $\Pr[g(\mathbf{x}', \mathbf{y}') \neq z_m, (\mathbf{x}', \mathbf{y}') \in C_{\ell_1}^x \times C_{\ell_2}^y] \leq 6\epsilon' \cdot \Pr[(\mathbf{x}', \mathbf{y}') \in C_{\ell_1}^x \times C_{\ell_2}^y]$ .

**Proof.** We prove that  $\ell_1, \ell_2$  exist via the probabilistic method. Let  $\ell_1$  and  $\ell_2$  be the buckets of  $\mathbf{x}'$  and  $\mathbf{y}'$ , respectively. Thus  $\Pr[\ell_1 = \ell] = \Pr[\mathbf{x}' \in C_\ell^x]$  and  $\Pr[\ell_2 = \ell] = \Pr[\mathbf{y}' \in C_\ell^y]$ .

Let  $B_1, B_2 \subseteq L' = \{1, \dots, 2^{7c'}\}$  be sets of indices of small probability, i.e., for  $i \in \{1, 2\}$

$$B_i = \left\{ \ell \in L' \mid \Pr[\ell_i = \ell] \leq \frac{1}{6 \cdot 2^{7c'}} \right\}.$$

We will prove that with high probability we have  $2^{7c'} \geq \ell_1 > 1$  and  $\ell_1 \notin B_1$ . The proof for  $\ell_2$  is analogous.

$$\Pr[\ell_1 = 1] = \Pr[\mathbf{x}' \in C_1^x] = \sum_{x \in C_1^x} \mu'_z(x) \leq \frac{\sum_{x \in C_1^x} \mu(x)}{2^{c'}} \leq \frac{1}{2^{c'}}$$

By Lemma 13, we get  $\Pr[\ell_1 > 2^{7c'}] = \Pr[\mu'_z(\mathbf{x}') \geq 2^{6c'} \cdot \mu(\mathbf{x}')] \leq \frac{1}{6}$ . There is only small probability that  $\ell_1$  is in  $B_1$ .

$$\Pr[\ell_1 \in B_1] = \sum_{\ell \in B_1} \Pr[\ell_1 = \ell] \leq \frac{|L'|}{6 \cdot 2^{7c'}} = \frac{1}{6}$$

Thus, we have that  $\ell_i \in B_i$  or  $\ell_i = 1$  or  $\ell_i > 2^{7c'}$  with probability at most  $\frac{2}{3} + \frac{2}{2^{c'}}$ .

## 20:12 Lower Bounds for Semi-adaptive Data Structures via Corruption

By Lemma 12, we have that  $\Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m] \leq \varepsilon'$ . By expanding the probability and by Markov inequality we will now get the last inequality for  $C_{\ell_1}^x$  and  $C_{\ell_2}^y$ . Let

$$e(\ell_1, \ell_2) = \Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m \mid \mathbf{x}' \in C_{\ell_1}^x, \mathbf{y}' \in C_{\ell_2}^y].$$

We will prove there is  $\ell_1$  and  $\ell_2$  such that  $e(\ell_1, \ell_2) \leq 6\varepsilon'$ . This is equivalent to the third bound of the lemma. We have:  $\varepsilon' \geq \Pr[g(\mathbf{x}', \mathbf{y}') \neq^* z_m] = \mathbb{E}[e(\ell_1, \ell_2)]$  and thus, by Markov,  $\Pr[e(\ell_1, \ell_2) > 6\varepsilon'] \leq \frac{1}{6}$ . By a union bound we conclude that there must exist  $1 < \ell_1, \ell_2 \leq 2^{7c'}$  such that  $\Pr[\ell_1 = \ell_1], \Pr[\ell_2 = \ell_2] \geq \frac{1}{6 \cdot 2^{7c'}}$  and  $e(\ell_1, \ell_2) \leq 6\varepsilon'$ . ◀

As a corollary we will prove that the rectangle  $C_{\ell_1}^x \times C_{\ell_2}^y$  (given by the previous lemma) is a good rectangle under the original distribution  $\mu$ . We remark that the proof of the following corollary is the only place where we use the fact that  $\mathbf{x}$  and  $\mathbf{y}$  are independent.

► **Corollary 15.** *There exists a rectangle  $R \subseteq S_x \times S_y$  such that*

1.  $\Pr[(\mathbf{x}, \mathbf{y}) \in R] \geq \frac{1}{36 \cdot 2^{26c'}}$ .
2.  $\Pr[g(\mathbf{x}, \mathbf{y}) \neq^* z_m, (\mathbf{x}, \mathbf{y}) \in R] \leq 24\varepsilon' \cdot \Pr[(\mathbf{x}, \mathbf{y}) \in R]$ .

**Proof.** Let  $R = C_{\ell_1}^x \times C_{\ell_2}^y$  where  $C_{\ell_1}^x$  and  $C_{\ell_2}^y$  are buckets given by Lemma 14. By Lemma 14, we get

$$\frac{1}{6 \cdot 2^{7c'}} \leq \Pr[\mathbf{x}' \in C_{\ell_1}^x] = \sum_{x \in C_{\ell_1}^x} \mu'_z(x) \leq \sum_{x \in C_{\ell_1}^x} \frac{\ell_1 \cdot \mu(x)}{2^{c'}} = \Pr[\mathbf{x} \in C_{\ell_1}^x] \cdot \frac{\ell_1}{2^{c'}}.$$

By rearranging we get

$$\Pr[\mathbf{x} \in C_{\ell_1}^x] \geq \frac{2^{c'}}{6\ell_1 \cdot 2^{7c'}} \geq \frac{1}{6 \cdot 2^{13c'}}$$

The bound for  $\Pr[\mathbf{y} \in C_{\ell_2}^y]$  is analogous, thus we have  $\Pr[(\mathbf{x}, \mathbf{y}) \in R] \geq \frac{1}{36 \cdot 2^{26c'}}$ . (Here and below, we crucially use the fact that  $\mathbf{x}, \mathbf{y}$  are given by a product distribution.) Now we prove the second bound for  $R$ . Let  $B = \{(x, y) \in R : g(x, y) \neq z_m, g(x, y) \neq^* \}$ .

$$\begin{aligned} 6\varepsilon' \cdot \Pr[(\mathbf{x}, \mathbf{y}) \in R] \cdot \frac{\ell_1 \ell_2}{2^{2c'}} &\geq 6\varepsilon' \cdot \Pr[(\mathbf{x}', \mathbf{y}') \in R] && \text{by definition of buckets} \\ &\geq \Pr[(\mathbf{x}', \mathbf{y}') \in B] && \text{by Lemma 14} \\ &\geq \Pr[(\mathbf{x}, \mathbf{y}) \in B] \cdot \frac{(\ell_1 - 1)(\ell_2 - 1)}{2^{2c'}} && \text{by definition of buckets} \end{aligned}$$

Thus, by rearranging we get

$$\Pr[(\mathbf{x}, \mathbf{y}) \in B] \leq 6\varepsilon' \cdot \Pr[(\mathbf{x}, \mathbf{y}) \in R] \cdot \frac{\ell_1 \ell_2}{(\ell_1 - 1)(\ell_2 - 1)} \leq 24\varepsilon' \cdot \Pr[(\mathbf{x}, \mathbf{y}) \in R],$$

as  $\frac{\ell_1 \ell_2}{(\ell_1 - 1)(\ell_2 - 1)} \leq 4$  for  $\ell_1, \ell_2 > 1$  by Lemma 14. ◀

**Proof of Theorem 5.** Suppose that  $t_u \cdot n \leq o(k/w)$ . Let  $R$  be the rectangle given by Corollary 15. It holds that the rectangle  $R$  is  $24\varepsilon'$ -error  $b$ -monochromatic for  $g$  under  $\mu$ . Therefore, for the function  $g$  holds that

$$\text{mono}_{\mu}^{24\varepsilon'}(g) \geq \Pr[(\mathbf{x}, \mathbf{y}) \in R] \geq \frac{1}{36 \cdot 2^{26c'}}. \quad (1)$$

We need to argue that  $\varepsilon'$  is  $O(\varepsilon/\alpha)$ . By definition,

$$\varepsilon' = \frac{5}{\alpha - \varepsilon} \cdot \varepsilon + \delta.$$

We recall that

$$\delta = O\left(\sqrt{\frac{t_u \cdot n \cdot w}{p}}\right) \leq \sqrt{\frac{o(k)}{p}}.$$

Thus, we can set  $p$  to be large enough so that  $\delta$  is smaller than arbitrary constant and still  $p \leq o(k)$ . By the assumption we have  $2\varepsilon < \alpha$ . Thus,  $\frac{\varepsilon}{\alpha - \varepsilon} \leq \frac{2\varepsilon}{\alpha}$  and we conclude that  $\varepsilon'$  is  $O(\varepsilon/\alpha)$ . Since  $c' = O\left(\frac{t_q \cdot w}{\alpha \cdot (1 - \varepsilon)}\right) = O\left(\frac{t_q \cdot w}{\alpha}\right)$ , we get the result by rearranging Inequality (1). ◀

## 4 Applications

In this section we apply Theorem 5 to derive lower bounds for several explicit functions – Inner Product (IP), Disjointness (DISJ), Gap Orthogonality (ORT) and Gap Hamming Distance (GHD):

$$\begin{aligned} \text{IP}(x, y) &= \sum_{i \in [n]} x_i \cdot y_i \pmod{2}, \\ \text{GHD}_n(x, y) &= \begin{cases} 1 & \text{if } \Delta_H(x, y) \geq \frac{n}{2} + \sqrt{n}, \\ 0 & \text{if } \Delta_H(x, y) \leq \frac{n}{2} - \sqrt{n}. \end{cases} \end{aligned}$$

The function  $\Delta_H$  is the Hamming Distance of two strings, i.e.,  $\Delta_H(x, y)$  is a number of indices  $i \in [n]$  such that  $x_i \neq y_i$ . For  $\text{IP}_{\mathbb{R}}(x, y) = \sum_{i \in [n]} (-1)^{x_i + y_i}$  we define

$$\text{ORT}_{n,d}(x, y) = \begin{cases} 1 & \text{if } |\text{IP}_{\mathbb{R}}(x, y)| \geq 2d \cdot \sqrt{n} \\ 0 & \text{if } |\text{IP}_{\mathbb{R}}(x, y)| \leq d \cdot \sqrt{n}. \end{cases}$$

The standard value for  $d$  is 1, thus we denote  $\text{ORT}_n = \text{ORT}_{n,1}$ . Note that  $\Delta_H(x, y) = \frac{n - \text{IP}_{\mathbb{R}}(x, y)}{2}$  and  $\text{IP}_{\mathbb{R}}(x, y)$  is the Inner Product of  $x', y'$  over  $\mathbb{R}$  where  $x'$  and  $y'$  arise from  $x$  and  $y$  by replacing 0 by 1 and 1 by  $-1$ . We present previous results with bounds for measures of interest under hard distributions.

► **Theorem 16** ([12]). *Let  $\mu_1$  be a uniform distribution on  $\{0, 1\}^n \times \{0, 1\}^n$ . Then,*

$$\text{disc}_{\mu_1}(\text{IP}) \leq \frac{1}{2^{n/2}}.$$

► **Theorem 17** (Babai et al. [2]). *Let  $\rho < 1/100$  and  $\mu_2$  be a uniform distribution over  $S \times S$ , where  $S$  consists of  $n$ -bit strings containing exactly  $\sqrt{n}$  1's. Then,*

$$\text{mono}_{\mu_2}^{\rho}(\text{DISJ}) \leq \frac{1}{2^{\Omega(\sqrt{n})}}.$$

Sherstov [20] provided a lower bound of communication complexity of GHD by lower bound of corruption bound of  $\text{ORT}_{n, \frac{1}{8}}$  following by reduction to GHD.

► **Theorem 18** (Sherstov [20]). *Let  $\rho > 0$  be sufficiently small and  $\mu_3$  be a uniform distribution over  $\{0, 1\}^n \times \{0, 1\}^n$ . Then,*

$$\text{cb}_{\mu_3}^{\rho}(\text{ORT}_{n, \frac{1}{8}}) \geq \rho \cdot n.$$

## 20:14 Lower Bounds for Semi-adaptive Data Structures via Corruption

By this theorem and Theorem 5 we get a lower bound for data structures for  $\text{ORT}_{n, \frac{1}{8}}$ . By reductions used by Sherstov [20] we also get a lower bounds for ORT and GHD.

$$\begin{aligned} \text{ORT}_{n, \frac{1}{8}}(x, y) &= \text{ORT}_{64n}(x^{64}, y^{64}) \\ \text{ORT}_n(x, y) &= \text{GHD}_{10n+15\sqrt{n}}(x^{10}1^{15\sqrt{n}}, y^{10}0^{15\sqrt{n}}) \\ &\quad \wedge \neg\text{GHD}_{10n+15\sqrt{n}}(x^{10}0^{15\sqrt{n}}, y^{10}0^{15\sqrt{n}}) \end{aligned}$$

Where  $s^i$  denote  $i$  copies of  $s$  concatenated together. Let  $D$  be a semi-adaptive random scheme for the multiphase problem of the presented functions with sufficiently small error probability. By the theorems presented in this section and by Theorem 5, we can derive the following lower bounds for  $t_q \cdot w$ , assuming that  $t_u \cdot n \leq o(k/w)$ .

Function $f$	Ballancedness of the hard distribution	Lower bound of $t_q \cdot w$
IP	$\frac{1}{2}$	$\Omega(n)$
DISJ	$\sim \frac{1}{e}$	$\Omega(\sqrt{n})$
$\text{ORT}_n$	$\Theta(1)$	$\Omega(n)$
$\text{GHD}_n$	N/A (lower-bound is via reduction)	$\Omega(n)$

---

### References

- 1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th FOCS*, pages 434–443, 2014.
- 2 Laszlo Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proceedings of the 27th FOCS*, page 337–347, 1986.
- 3 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd FOCS*, page 209–218, 2002.
- 4 Paul Beame, Toniann Pitassi, Nathan Segerlind, and Avi Wigderson. A strong direct product theorem for corruption and the multiparty communication complexity of disjointness. *Computational Complexity*, 15(4):391–432, 2006.
- 5 Amit Chakrabarti, Ranganath Kondapally, and Zhenghui Wang. Information complexity versus corruption and applications to orthogonality and gap-hamming. In *Proceedings of the 16th RANDOM*, pages 483–494. Springer, 2012.
- 6 Arkadev Chattopadhyay, Jeff Edmonds, Faith Ellen, and Toniann Pitassi. A Little Advice Can Be Very Helpful. In *Proceedings of the 23rd SODA*, pages 615–625, 2012.
- 7 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.
- 8 Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st STOC*, pages 345–354, 1989.
- 9 Rahul Jain and Hartmut Klauck. The partition bound for classical communication complexity and query complexity. In *Proceedings of the 25th CCC*, page 247–258, 2010.
- 10 Bala Kalyanasundaram and Georg Schintger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal of Discrete Mathematics*, 5(4):545–557, 1992.
- 11 Young Kun Ko and Omri Weinstein. An Adaptive Step Toward the Multiphase Conjecture, 2019. [arXiv:1910.13543](https://arxiv.org/abs/1910.13543).
- 12 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996.
- 13 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the 44th STOC*, pages 85–94, 2012.

- 14 Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. *SIAM Journal on Computing*, 2020.
- 15 Mihai Pătraşcu and Erik D Demaine. Tight bounds for the partial-sums problem. In *Proceedings of the 15th SODA*, pages 20–29, 2004.
- 16 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd STOC*, pages 603–610, 2010.
- 17 Mihai Patrascu and Erik D Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.
- 18 Mihai Pătraşcu. Towards Polynomial Lower Bounds for Dynamic Problems. In *Proceedings of the 42nd STOC*, pages 603–610, 2010.
- 19 Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106:385–390, 1992.
- 20 Alexander A Sherstov. The communication complexity of gap hamming distance. *Theory of Computing*, 8(1):197–208, 2012.
- 21 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the 11th STOC*, pages 209–213, 1979.



# Stability-Preserving, Time-Efficient Mechanisms for School Choice in Two Rounds

**Karthik Gajulapalli**

Department of Computer Science, University of California Irvine, CA, US  
kgajulap@uci.edu

**James A. Liu**

K-Sky Limited, Hong Kong, Hong Kong  
james@k-sky.hk

**Tung Mai**

Adobe Research, San Jose, CA, US  
tumai@adobe.com

**Vijay V. Vazirani**

Department of Computer Science, University of California Irvine, CA, US  
vazirani@ics.uci.edu

---

## Abstract

We address the following dynamic version of the school choice question: a city, named City, admits students in two temporally-separated rounds, denoted  $\mathcal{R}_1$  and  $\mathcal{R}_2$ . In round  $\mathcal{R}_1$ , the capacity of each school is fixed and mechanism  $\mathcal{M}_1$  finds a student optimal stable matching. In round  $\mathcal{R}_2$ , certain parameters change, e.g., new students move into the City or the City is happy to allocate extra seats to specific schools. We study a number of Settings of this kind and give polynomial time algorithms for obtaining a stable matching for the new situations.

It is well established that switching the school of a student midway, unsynchronized with her classmates, can cause traumatic effects. This fact guides us to two types of results: the first simply disallows any re-allocations in round  $\mathcal{R}_2$ , and the second asks for a stable matching that minimizes the number of re-allocations. For the latter, we prove that the stable matchings which minimize the number of re-allocations form a sublattice of the lattice of stable matchings. Observations about incentive compatibility are woven into these results. We also give a third type of results, namely proofs of NP-hardness for a mechanism for round  $\mathcal{R}_2$  under certain settings.

**2012 ACM Subject Classification** Mathematics of computing → Discrete mathematics

**Keywords and phrases** stable matching, mechanism design, NP-Hardness

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.21

**Related Version** Full version: <https://arxiv.org/abs/1904.04431>.

## 1 Introduction

School choice is among the most consequential events in a child’s upbringing, whether it is admission to elementary, middle or high school, and hence has been accorded its due importance not only in the education literature but also in game theory and economics. In order to deal with the flaws in the practices of the day, the seminal paper of Abdulkadiroglu and Sonmez [3] formulated this as a mechanism design problem. This approach has been enormously successful, especially in large cities involving the admission of tens of thousands of students into hundreds of schools, e.g., see [2, 1, 4, 18], and today occupies a key place in the area of market design in economics, e.g., see [23, 21, 22, 13].

Once the basic game-theoretic issues in school choice were adequately addressed, researchers turned attention to the next level of questions. In this vein, in a recent paper, Feigenbaum et. al. [12] remarked, “However, most models considered in this literature are essentially



© Karthik Gajulapalli, James A. Liu, Tung Mai, and Vijay V. Vazirani;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 21; pp. 21:1–21:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

static. Incorporating dynamic considerations in designing assignment mechanisms ... is an important aspect that has only recently started to be addressed.”

Our paper deals with precisely this. We define several settings for school choice in which an instance is made available in the first round  $\mathcal{R}_1$  and at a later time, in the second round  $\mathcal{R}_2$ , some of the parameters change. Each setting asks for a pair of mechanisms,  $(\mathcal{M}_1, \mathcal{M}_2)$  for finding matchings of students to schools in these two rounds. All our settings insist that the matchings found in both rounds are stable. It will be convenient to classify our results into three types. In Type A and B, both mechanisms  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are required to run in polynomial time.

1. **Type A:** Mechanism  $\mathcal{M}_2$  is disallowed to reassign the school of any student matched by  $\mathcal{M}_1$ . We present two settings, **A1** and **A2**.
2. **Type B:** Mechanism  $\mathcal{M}_2$  is allowed to reassign the school of students matched by  $\mathcal{M}_1$ ; however, it needs to (provably) minimize the number of such reassignments. We present two settings, **B1** and **B2**.
3. **Type C:** These are NP-hardness results – of mechanism  $\mathcal{M}_2$  for four problems and of a fifth problem, which involves only one round.

### 1.1 Our model and its justification

Our solutions to Type A and B results will strictly adhere to the following tenets; we justify them below.

1. **Tenet 1:** All matchings produced by our mechanisms need to be stable.
2. **Tenet 2:** In Type A results, mechanism  $\mathcal{M}_2$  is disallowed to reassign the school of any student matched by  $\mathcal{M}_1$ , and in Type B results,  $\mathcal{M}_2$  must provably minimize the number of such reassignments.
3. **Tenet 3:** We want all our mechanisms to run in polynomial time.

The use of the classic Gale-Shapley [14] Deferred Acceptance Algorithm has emerged as a method of choice in the literature. Our mechanisms also use this algorithm. Stability comes with key advantages: First, no student and school, who are not matched to each other, will have the incentive to go outside the mechanism to strike a deal. Second, it eliminates *justified envy*, i.e., the following situation cannot arise: there is a student  $s_i$  who prefers another student  $s_j$ 's school assignment, say  $h_k$ , while being fully aware that  $h_k$  preferred her to  $s_j$ .

Switching the school of a student midway, unsynchronized with her classmates – such as when the entire class moves from elementary to middle or from middle to high school – is well-known to cause traumatic effects, e.g., see [15]. It is for these reasons that in Type A results, mechanism  $\mathcal{M}_2$  is disallowed to reassign the school of any student matched by  $\mathcal{M}_1$  and in Type B results,  $\mathcal{M}_2$  must provably minimize the number of such reassignments. For Type A results, we say that  $\mathcal{M}_2$  extends  $M$  to a stable matching  $M'$ . For Type B results, we say that  $\mathcal{M}_2$  computes a minimum stable re-allocation  $M'$  of  $M$ .

The strongest notion of incentive compatibility for a mechanism is *dominant strategy incentive compatible (DSIC)*, for students. This entails that regardless of the preferences reported by other students, a student can do no better than report her true preference list, i.e., truth-telling is a dominant strategy for all students. This immediately simplifies the task of students and their parents, since they don't need to waste any effort trying to game the system. Furthermore, if students are forced to adjust their choices in an attempt to gain a better matching, the mechanism, dealing with choices reported to it, may be forced to make matches that are suboptimal for students as well as schools.

Gale and Shapley [14] proved that if the Differed Acceptance Algorithm is run with students proposing, it will yield a *student-optimal matching*, i.e., each student will get the best possible school, according to her preference list, among all stable matchings. However, this matching may be extremely unfavorable to an individual student – it may be matched to a school which is very low on her preference list, giving her incentive to cheat, i.e., provide a false preference list, in order to get a better matching. Almost two decades after the Gale-Shapley result, Dubins and Freedman [8] proved, via a highly non-trivial analysis, that this algorithm is DSIC for students. This ground-breaking result opened up the Gale-Shapley algorithm to a host of highly consequential applications, including school choice.

In all of our results of Type A and B, mechanism  $\mathcal{M}_1$  finds a student-optimal stable matching using the Gale-Shapley Differed Acceptance Algorithm and is therefore DSIC for students. For Setting B2 we provide a mechanism for round  $\mathcal{R}_2$  that is DSIC. However, our mechanisms for round  $\mathcal{R}_2$  for the remaining three settings do not achieve this. Our main open problem is to fix this. For completeness, and in order to motivate this open problem, we discuss incentive-compatibility for each of these setting in Section 6.

It is well known that the set of Stable Matchings of a given instance forms a finite distributive lattice [16]. By orienting the underlying partial order of this lattice appropriately, the student-optimal stable matching can be made the top element of this lattice and the school optimal matching the bottom element. For both Settings of Type B, we show that the set of minimum stable re-allocations form a sublattice of this lattice. We provide polynomial-time mechanisms for computing the top and bottom elements of this sublattice. For Setting B1, we show that the top of the sublattice is also the top of the whole lattice, i.e., it is the student-optimal stable matching; this is crucial for showing DSIC for B1.

### 1.1.1 Type A and B settings

The four settings involve the admission of students of a city, named City, into schools; the preference lists of both students and schools are provided to the mechanisms.  $\mathcal{M}_1$  computes a student-optimal stable matching,  $M$ , over all the participants in  $\mathcal{R}_1$ . In  $\mathcal{R}_2$  some of the parameters over which  $M$  was defined are updated.  $\mathcal{M}_2$  then modifies the matching  $M$  to produce a new matching  $M'$  that is stable over the new parameters defined in  $\mathcal{R}_2$ .

For Settings of Type A, in round  $\mathcal{R}_1$ , the capacity of each school is fixed but in round  $\mathcal{R}_2$ , the City is happy to allocate extra seats to specific schools per the recommendation of mechanism  $\mathcal{M}_2$ , which in turn has to meet specified requirements imposed by the City. Let  $L$  be the set of *left-over students*, those who could not be admitted in round  $\mathcal{R}_1$ .

In round  $\mathcal{R}_2$  of Setting A1, the problem is to maximize the number of students admitted from  $L$ , by extending  $M$  in a stability-preserving manner. In Setting A2, a set  $N$  of *new students* also arrive from other cities and their preference lists are revealed to  $\mathcal{M}_2$ . The requirement now is to admit as few students as possible from  $N$  and subject to that, as many as possible from  $L$ , again in a stability-preserving manner. Finally, we give a procedure that outputs all possible stability-preserving extensions of a given stable matching (which may be exponentially many) with polynomial delay.

For Settings of Type B, the capacity of each school is fixed in  $\mathcal{R}_1$ , but in  $\mathcal{R}_2$  the City has to deal with the arrival of new students and new schools. This could lead the matching found by  $\mathcal{M}_1$  to no longer be stable.

In round  $\mathcal{R}_2$  of Setting B1, a set  $N$  of new students arrive and their preference lists are revealed to  $\mathcal{M}_2$ . The capacity of schools remain unchanged and the problem is to find a matching,  $M'$  that is stable under the arrival of new students which minimizes the number of students who are assigned to a different school in  $M'$ . In Setting B2, a set  $H'$  of new schools

arrive and the City allows the capacities of the original schools to increase. The preference lists of the students are updated to reflect these new schools, we again require that  $\mathcal{M}_2$  compute a new stable matching,  $M'$  over the updated preference lists that minimizes the number of students who get matched to a different school in  $M'$ .

## 1.2 Related work

Besides the references pointed out above on school choice, in this section, we will concentrate on recent work on dynamic matching markets, especially those pertaining to school choice. Feigenbaum et. al. [12] study the following issue that arises in NYC public high schools, which admits over 80,000 students annually: after the initial centralized allocation, about 10% of the students choose not attend the school allocated to them, instead going to private or charter schools. To deal with this, [12] give a two-round solution which maintains truthfulness and efficiency and minimizes the movement of students between schools.

An interesting phenomena that has been observed in matching markets is *unraveling*, under which matches are made early to beat the competition, even though it leads to inefficiencies due to unavailability of full information. A classic case, indeed one that motivated the formation of centralized clearing houses, is that of the market for medical interns in which contracts for interns were signed two years before the future interns would even graduate [19]. A theoretical explanation of this phenomena was recently provided by [11].

[17] point out that stable pairings may not necessarily last forever, e.g., a student may switch from private to public school or a married couple may divorce. They study dynamic, multi-period, bilateral matching markets and they define and identify sufficient conditions for the existence of a dynamically stable matching.

[7] develops a notion of stability that applies in markets where matching opportunities arrive over time, much like the seats in our work. One of the things shown in this paper is that agents' incentive to wait for better matching opportunities can make achieving stability very difficult. Indeed, the notion of dynamic stability given in this paper is a necessary condition which a matching must satisfy in order that agents do not find it profitable to game a mechanism by showing up in later rounds.

A number of recent papers [24, 6, 5, 10] consider the consequences of having a mechanism that repeats the Gale-Shapley Deferred Acceptance algorithm multiple times, similar to our work. Note that Deferred Acceptance is not *consistent* in that if one runs it, then removes some agents and their assignments, and runs it again on the remaining agents, one does not obtain the same assignment restricted to the left-over agents. In these papers, the authors show that there is room for manipulation by submitting empty lists in the first round. However, unlike our model in which changes are introduced in round  $\mathcal{R}_2$ , in all these papers, there is nothing that motivates running Deferred Acceptance twice, namely no arrivals of new students, no change in capacities, no changes in preferences, etc.

## 1.3 Overview of structural and algorithmic ideas

The main idea for obtaining a stability-preserving mechanism in round  $\mathcal{R}_2$  for Settings  $A1$  and  $A2$  lies in the notion of a *barrier* which ensures that students admitted in  $\mathcal{R}_2$  do not form blocking pairs. A crucial issue is to place barriers optimally to ensure that the number of students admitted is optimized (minimized or maximized) appropriately.

The algorithm for enumerating stable extensions of a stable matching, given in Section 4.3, relies heavily on the fundamental structural property of stable matchings. Enumerated matchings are extended by only one student in an iteration. At each step, the algorithm

finds all such feasible extensions by one student in a way such that there must be at least one feasible assignment, for any student, at each step. This assurance is crucial in guaranteeing that the delay between any two enumerated matchings is polynomial.

For Settings B1 and B2, the mechanism proceeds by iteratively resolving blocking pairs. Structurally, we show that the set of all minimum stable re-allocations forms a sublattice of the stable matching lattice. Our analysis relies on the fact that the set of students who are assigned to a different school in round  $\mathcal{R}_2$  cannot be matched to their original school in any minimum stable re-allocation. This lets us divide the set of students into two groups, students matched to the same school (fixed students), and students matched to different schools (moving students). We then construct a smaller stable matching instance,  $I$ , over the set of moving students. By appropriately placing barriers for each student and school in  $I$  we can ensure that the union of any stable matching in  $I$  and the matching restricted to the fixed students will also be stable. This stable matching is a minimum stable re-allocation and defines a bijection between the set of minimum stable re-allocations and set of stable matchings in  $I$ , we exploit the lattice structure of the latter.

## 2 Preliminaries

### 2.1 The stable matching problem for school choice

The stable matching problem takes as input a set  $H = \{h_1, h_2, \dots, h_m\}$  of  $m$  public schools and a set  $S = \{s_1, s_2, \dots, s_n\}$  of  $n$  students who are seeking admission to the schools. Each school  $h_j \in H$  has an integer-valued *capacity*,  $c(j)$ , stating the maximum number of students that can be assigned to it. If  $h_j$  is assigned  $c(j)$  students, we will say that  $h_j$  is *filled*, and otherwise it is *under-filled*.

Each student  $s_i \in S$  has a strict and complete preference list,  $l(s_i)$ , over  $H \cup \{\emptyset\}$ . If  $s_i$  prefers  $\emptyset$  to  $h_j$ , then it prefers remaining unassigned rather than being assigned to school  $h_j$ . We will assume that the list  $l(s_i)$  is ordered by decreasing preferences. Therefore, if  $s_i$  prefers  $h_j$  to  $h_k$ , we can equivalently say that  $h_j$  appears *before*  $h_k$  or  $h_k$  appears *after*  $h_j$  on  $s_i$ 's preference list. Clearly, the order among the schools occurring after  $\emptyset$  on  $s_i$ 's list is immaterial, since  $s_i$  prefers remaining unassigned rather than being assigned to any one of them. Similarly, each school  $h_j \in H$  has a strict and complete preference list,  $l(h_j)$ , over  $S \cup \{\emptyset\}$ . Once again, for each student  $s_i$  occurring after  $\emptyset$ ,  $h_j$  prefers remaining under-filled rather than admitting  $s_i$ , and the order among these students is of no consequence.

Given a set of schools,  $H' \subseteq H$ , by the *best school for  $s_i$  in  $H'$*  we mean the school that  $s_i$  prefers the most among the schools in  $H'$ . Similarly, given a set of students,  $S' \subseteq S$ , by the *best student for  $h_j$  in  $S'$*  we mean the student whom  $h_j$  prefers the most among the students in  $S'$ .

A *matching*  $M$  is a function,  $M : S \rightarrow H \cup \{\emptyset\}$  such that if  $M(s_i) = h_j$  then it must be the case that  $s_i$  prefers  $h_j$  to  $\emptyset$  and  $h_j$  prefers  $s_i$  to  $\emptyset$ ; if so, we say that student  $s_i$  is *assigned to school  $h_j$* . If  $M(s_i) = \emptyset$ , then  $s_i$  is not assigned to any school. The matching  $M$  also has to ensure that the number of students assigned to each school  $h_j$  is at most  $c(j)$ .

For a matching  $M$ , a student-school pair  $(s_i, h_j)$  is said to be a *blocking pair* if  $s_i$  is not assigned to  $h_j$ ,  $s_i$  prefers  $h_j$  to  $M(s_i)$  and one of the following conditions holds:

1.  $h_j$  prefers  $s_i$  to one of the students assigned to  $h_j$ , or
2.  $h_j$  is under-filled and  $h_j$  prefers  $s_i$  to  $\emptyset$ .

The blocking pair is said to be *type 1* (*type 2*) if the first (second) condition holds. A matching  $M$  is said to be *stable* if there is no blocking pair for it.

► **Theorem 1** (Rural Hospitals Theorem [20]).

1. Over all the stable matchings of the given instance: the set of matched students is the same and the number of students matched to each school is also the same.
2. Assume that school  $h$  is not matched to capacity in a stable matching. Then, the set of students matched to  $h$  is the same over all stable matchings.

## 2.2 The Stable Matching Lattice

► **Definition 2.** A Lattice  $\mathcal{L} = (S, \succeq)$ , is defined over a finite set  $S$ , and a partial order  $\succeq$ , if for every pair of elements  $a, b \in S$ , there exists a unique least upperbound and a unique greatest lowerbound. We call the least upperbound the join of  $a$  and  $b$  and denote it by  $a \vee b$ , and analogously call the least lowerbound the meet of  $a$  and  $b$  and denote it by  $a \wedge b$

► **Definition 3.** Let  $SM$  denote the set of stable matchings over given instance  $(S, H, c)$ , then for two stable matchings  $M, M' \subseteq SM$ ,  $M \succeq M'$ , if and only if  $\forall s_i \in S$ ,  $s_i$  weakly prefers  $M(s_i)$  to  $M'(s_i)$

Given two stable matchings  $M$  and  $M'$  consider two new maps  $M_U$  and  $M_L$ , defined as follows:

- $M_U(s_i) = \max(M(s_i), M'(s_i))$
- $M_L(s_i) = \min(M(s_i), M'(s_i))$

where  $\max$  is the partner  $s_i$  weakly prefers between  $M$  and  $M'$ , and  $\min$  is the complement of  $\max$ .

► **Theorem 4** ([16]). The set of stable matchings  $(SM, \succeq)$  characterizes a finite distributive lattice. Moreover  $M_L, M_U$  represent the meet and join of any two stable matchings in the lattice.

## 3 Our Results for the Four Settings

In round  $\mathcal{R}_1$ , the setup defined in Section 2.1 prevails and  $\mathcal{M}_1$  simply computes the student-optimal stable matching respecting the capacity of each school, namely  $c(j)$  for  $h_j$ . Let this matching be denoted by  $M$ ,  $S_M \subseteq S$  be the set of students assigned to schools by  $M$  and  $L = (S - S_M)$  be the set of *left-over students*. As shown in [9],  $\mathcal{M}_1$  is DSIC for students.

For Settings of Type A, in round  $\mathcal{R}_2$ , the City has decided to extend matching  $M$  in a stable manner without any restrictions on extra capacity added to each school.

For Settings of Type B, in round  $\mathcal{R}_2$  a change is made to the sets of participants, which may cause  $M$  to no longer be a valid or stable matching.  $\mathcal{M}_2$  then updates  $M$  to  $M'$  in order to ensure a stable matching. By allowing *updates*, we let some students in  $M$  get unmatched in  $M'$ , or get matched to different schools. The City would like to minimize the number of students who would have to change schools, or no longer be matched to a school, in going from  $M$  to  $M'$ . We call  $M'$  a **minimum stable re-allocation** of  $M$ . Formally,  $M'$  is a minimum stable re-allocation of  $M$  if  $M'$  is a stable matching over all participants and the number of students  $s_i \in S_M$  where  $M(s_i) \neq M'(s_i)$  is minimized.

### 3.1 Type A and B Settings

**Setting A1.** In this setting, in round  $\mathcal{R}_2$ , the City wants to admit as many students from  $L$  as possible in a stability-preserving manner. We will call this problem  $Max_L$ . We will prove the following:

► **Theorem 5.** *There is a polynomial time mechanism  $\mathcal{M}_2$  that extends matching  $M$  to  $M'$  so that  $M'$  is stable w.r.t. students  $S$  and schools  $H$ . Furthermore,  $\mathcal{M}_2$  yields the largest matching that can be obtained by a mechanism satisfying the stated conditions.*

Let  $k$  be the maximum number of students that can be added from  $L$ , as per Theorem 5. Next, suppose that the City can only afford to add  $k' < k$  extra seats. We show in Section 4.1 how this can be achieved while maintaining all the properties stated in Theorem 5.

**Setting A2.** In this setting, in round  $\mathcal{R}_2$ , in addition to the leftover set  $L$ , a set  $N$  of *new students* arrive from other cities and their preference lists are revealed to mechanism  $\mathcal{M}_2$ . Additionally, the schools also update their preference lists to include the new students. In this setting, the City wants to give preference to students who were not matched in round  $\mathcal{R}_1$ , i.e.,  $L$ , over the new students,  $N$ . Thus it seeks the subset of  $N$  that *must* be admitted to avoid blocking pairs and subject to that, maximize the subset of  $L$  that can be added, again in a stability-preserving manner. We will call this problem  $Min_N Max_L$ . We will prove the following:

- **Theorem 6.** *There is a polynomial time mechanism  $\mathcal{M}_2$  that accomplishes the following:*
1. *It finds smallest subset  $N' \subseteq N$  with which the current matching  $M$  needs to be extended in a stability-preserving manner.*
  2. *Subject to the previous extension, it finds the largest subset  $L' \subseteq L$  with which the matching can be extended further in a stability-preserving manner.*

**Setting B1.** In this setting, a set  $N$  of *new students* arrive from other cities in round  $\mathcal{R}_2$ . The preference lists of schools are also updated to include students in  $N$ , though their relative preferences between students in  $S \cup \{\emptyset\}$  are unchanged. The City wants to find a stable matching over students  $S \cup N$  and schools  $H$  that minimizes the number of students who are re-allocated from their original school in  $M$ .

► **Theorem 7.** *There is a polynomial time mechanism  $\mathcal{M}_2$  that finds a minimum stable reallocation with respect to Round  $\mathcal{R}_1$  matching  $M$ , students  $S \cup N$ , and schools  $H$ .*

**Setting B2.** In this setting, the City has some new schools  $H'$  that have opened up in  $\mathcal{R}_2$ . The preference lists of students are updated to include schools in  $H'$ , though their relative preferences between schools in  $H \cup \{\emptyset\}$  are unchanged. The City also allows schools in  $H$  to increase their capacity in round  $\mathcal{R}_2$ . The City wants to find a stable matching over students  $S$  and schools  $H \cup H'$  that minimizes the number of students who are re-allocated from their original school in  $M$ .

► **Theorem 8.** *There is a polynomial time mechanism  $\mathcal{M}_2$  that finds a minimum stable reallocation with respect to Round  $\mathcal{R}_1$  matching  $M$ , students  $S$ , and schools  $H \cup H'$ .*

## 4 Mechanisms for Type A Settings

### 4.1 Setting A1

We will first characterize situations under which a matching is not stable, i.e., admits a blocking pair. This characterization will be used for proving stability of matchings constructed in round  $\mathcal{R}_2$ . For this purpose, assume that  $M$  is an arbitrary matching, not necessarily stable nor related to the matching computed in round  $\mathcal{R}_1$ . For each school  $h_j \in H$ , define the *least preferred student assigned to  $h_j$* , denoted  $LPS\text{-Assigned}(h_j)$ , to be the student whom  $h_j$  prefers the least among the students that are assigned to  $h_j$ .

$Max_L(M, L)$ :

**Input:** Stable matching  $M$  and set  $L$ .

**Output:** Stable,  $Max_L$  extension of  $M$ .

1.  $\forall s_i \in S_M : M'(s_i) \leftarrow M(s_i)$
2.  $\forall h_j \in H : \text{Barrier}(h_j) \leftarrow \text{BS-Preferring}(h_j)$ .
3.  $L' \leftarrow \{s_i \in L \mid \exists h_j \text{ s.t. } s_i \text{ appears before } \text{Barrier}(h_j) \text{ in } l(h_j),$   
and  $h_j \text{ appears before } \emptyset \text{ in } l(s_i)\}$ .
4.  $\forall s_i \in L' : \text{Feasible-Schools}(s_i) \leftarrow \{h_j \mid s_i \text{ appears before } \text{Barrier}(h_j) \text{ in } l(h_j)\}$ .
5.  $\forall s_i \in L' : M'(s_i) \leftarrow \text{Best school for } s_i \text{ in Feasible-Schools}(s_i)$ .
6.  $\forall s_i \in (L - L') : M'(s_i) \leftarrow \emptyset$ .
7. Return  $M'$ .

■ **Figure 1** Mechanism for round  $\mathcal{R}_2$  for problem  $Max_L$  in Setting A1.

Next, for each student  $s_i \in S_M$ , define the *set of schools preferred by  $s_i$* , denoted Preferred-Schools( $s_i$ ) by  $\{h_j \mid s_i \text{ prefers } h_j \text{ to } M(s_i)\}$ ; note that  $M(s_i) = \emptyset$  is allowed in this definition. Further, for each school  $h_j \in H$ , define the *set of students that prefer  $h_j$  over the school they are assigned to*, denoted Preferring-Students( $h_j$ ) to be  $\{s_i \mid h_j \in \text{Preferred-Schools}(s_i)\}$ . Finally, define *best student preferring  $h_j$* , denoted BS-Preferring( $h_j$ ), to be the student whom  $h_j$  prefers the best in the set Preferring-Students( $h_j$ ). If Preferring-Students( $h_j$ ) =  $\emptyset$  then we will define BS-Preferring( $h_j$ ) =  $\emptyset$ ; in particular, this happens if  $h_j$  is under-filled.

The mechanism  $\mathcal{M}_2$  for round  $\mathcal{R}_2$  for  $Max_L$  in Setting A1 is given in Figure 1. Step 1 simply ensures that the matching found by  $\mathcal{M}_2$  extends the round  $\mathcal{R}_1$  matching. Step 2 defines the Barrier for each school to be BS-Preferring( $h_j$ ); observe that this could be  $\emptyset$ . Step 3 determines the set  $L' \subseteq L$  that can be assigned schools in a stability-preserving manner and Step 5 computes the school for each student in this subset.

For the problem of admitting fewer students, we give the following:

► **Proposition 9.** *Let  $k$  be the total number of students added from  $L$  in round  $\mathcal{R}_2$  in the previous theorem and let  $k' < k$ . There is a polynomial time mechanism  $\mathcal{M}_2$  that is stability-preserving, and extends matching  $M$  to  $M'$  so that  $|M'| - |M| = k'$ .*

## 4.2 Setting A2

The mechanism for round  $\mathcal{R}_2$  for  $Min_N Max_L$  in Setting A2 is given in Figure 2 provided in the appendix. Suppose there is a school  $h_j$ , student  $s_k \in S_M$  is assigned to it and there is a student  $s_i \in N$  such that  $h_j$  prefers  $s_i$  to  $s_k$ . Now, if  $s_i$  is kept unmatched,  $(s_i, h_j)$  will form a blocking pair of type 1. Next suppose  $h_j$  is under-filled and there is a student  $s_i \in N$  such that  $h_j$  and  $s_i$  prefer each other to  $\emptyset$ . This time, if  $s_i$  is kept unmatched,  $(s_i, h_j)$  will form a blocking pair of type 2. Motivated by this, for a student  $s_i$ , define the *set of schools forming blocking pairs with  $s_i$* , denoted Schools-FBPairs( $s_i$ ), to be:

$$\text{Schools-FBPairs}(s_i) = \{h_j \in H \mid h_j \text{ prefers } s_i \text{ to LPS-Assigned}(h_j), s_i \text{ prefers } h_j \text{ to } \emptyset\} \\ \cup \{h_j \in H \mid h_j \text{ is under-filled and } h_j \text{ and } s_i \text{ prefer each other to } \emptyset\}.$$

$Min_N Max_L(M, N, L)$ :

**Input:** Stable matching  $M$ , and sets  $N$  and  $L$ .

**Output:** Stable,  $Min_N Max_L$  extension of  $M$ .

1.  $\forall s_i \in S_M : M'(s_i) \leftarrow M(s_i)$
2.  $\forall h_j \in H : \text{Barrier1}(h_j) \leftarrow \text{BS-Preferring}(h_j)$ .
3.  $N' \leftarrow \{s_i \in N \mid \text{Schools-FBPairs}(s_i) \text{ is non-empty}\}$ .
4.  $\forall h_j \in H : \text{Barrier2}(h_j) \leftarrow \text{Best student for } h_j \text{ in } (N - N')$ .
5.  $\forall h_j \in H : \text{Barrier}(h_j) \leftarrow \text{Best student for } h_j \text{ in } \{\text{Barrier1}(h_j), \text{Barrier2}(h_j)\}$ .
6.  $L' \leftarrow \{s_i \in L \mid \exists h_j \text{ s.t. } s_i \text{ appears before } \text{Barrier}(h_j) \text{ in } l(h_j), \text{ and } h_j \text{ appears before } \emptyset \text{ in } l(s_i)\}$ .
7.  $\forall s_i \in (N' \cup L') : \text{Feasible-Schools}(s_i) \leftarrow \{h_j \mid s_i \text{ appears before } \text{Barrier}(h_j) \text{ in } l(h_j)\}$
8.  $\forall s_i \in (N' \cup L') : M'(s_i) \leftarrow \text{Best school for } s_i \text{ in } \text{Feasible-Schools}(s_i)$ .
9.  $\forall s_i \in ((L - L') \cup (N - N')) : M'(s_i) \leftarrow \emptyset$ .
10. Return  $M'$ .

■ **Figure 2** Mechanism for round  $\mathcal{R}_2$  for  $Min_N Max_L$  in Setting A2.

Therefore, all students in  $N'$ , computed in Step 3, need to be matched. Our mechanism keeps all students in  $N - N'$  unmatched, thereby minimizing the number of students matched from  $N$ .

We next describe the various barriers that need to be defined. The first one, defined in Step 2, plays the same role as that in Figure 1. As before, if  $h_j$  is under-filled,  $\text{Barrier1}(h_j) = \emptyset$ . If a student  $s_i \in (N' \cup L')$  appears after  $\text{Barrier1}(h_j)$  in  $l(h_j)$  and is assigned to  $h_j$ , then  $(\text{Barrier1}(h_j), h_j)$  will form a blocking pair. The second one,  $\text{Barrier2}(h_j)$  in  $(N - N')$  defined in Step 4. Again, if  $s_i \in (N' \cup L')$  appears after  $\text{Barrier2}(h_j)$  in  $l(h_j)$  and is assigned to  $h_j$ , then  $(\text{Barrier2}(h_j), h_j)$  will form a blocking pair. In step 5,  $\text{Barrier}(H_j)$  is defined to be the more stringent of these two barriers.

The final question is which school should  $s_i \in N'$  be matched to? One possibility is to compute for each student  $s_i$  the set

$$T(s_i) = \{h_j \in H \mid \exists s_k \text{ s.t. } M(s_k) = h_j, h_j \text{ prefers } s_i \text{ to } s_k, \text{ and } s_i \text{ prefers } h_j \text{ to } \emptyset\},$$

and match  $s_i$  to her best school in  $T(s_i)$ .

Assume that  $s_i$  is matched to  $h_j$  under this scheme. A blocking pair may arise as follows: Assume  $s_i$  prefers school  $h_k$  to  $h_j$  (of course,  $h_k \notin T(s_i)$ ), some student  $s_l \in L'$  has been assigned to  $h_k$  and  $h_k$  prefers  $s_i$  to  $s_l$ . If so,  $(s_i, h_k)$  will form a blocking pair. One remedy is to redefine the barrier for  $h_k$  so  $s_l$  is not assigned to  $h_k$ . However, this will make the barrier more stringent and the resulting mechanism will, in general, match fewer students from  $L$  than our mechanism. The latter is as follows: simply match  $s_i$  to the best school which prefers her to the Barrier of that school.

► **Theorem 10.** *There is a polynomial time mechanism  $\mathcal{M}_2$  that finds the largest subset of  $(N \cup L)$  that can be matched to schools and added to the current matching while maintaining stability. This mechanism also solves  $Max_N Max_L$  and  $Max_L Max_N$ .*

```

STABLEEXTENSION( $M, c, N$ ):
Input: Stable matching  $M$ , capacity  $c$ , new students  $N = \{s_1, s_2 \dots s_k\}$ .
Output: Stable extensions of  $M$ , with polynomial delay.
 $M_0 \leftarrow M$ 
 $A_1 = \text{FEASIBLEASSIGNMENT}(M_0, c, s_1)$ 
For  $i_1$  in  $A_1$ :
     $M_1 \leftarrow$  Starting from  $M_0$ , match  $s_1$  to  $i_1$ .
     $A_2 = \text{FEASIBLEASSIGNMENT}(M_1, c, s_2)$ .
    For  $i_2$  in  $A_2$ :
         $\vdots$ 
         $A_k = \text{FEASIBLEASSIGNMENT}(M_{k-1}, c, s_k)$ .
        For  $i_k$  in  $A_k$ :
             $M_k \leftarrow$  Starting from  $M_{k-1}$ , match  $s_k$  to  $i_k$ .
            Enumerate  $M_k$ .

```

■ **Figure 3** Algorithm for enumerating stable extensions of  $M$ .

### 4.3 Enumeration of Stable Extensions

In this section we show how to enumerate all the possible stable extensions of a given stable matching with polynomial delay between any two enumerated matchings. Specifically, the algorithm takes as input a stable matching  $M$  from  $S$  to  $H$  satisfying capacity  $c$  and a set of new students  $N = \{s_1, s_2 \dots s_k\}$  that can be added to the schools. Here the preference lists of all schools and students are also given. The algorithm enumerates all solutions  $M'$  from  $S \cup N$  to  $H \cup \{\emptyset\}$  such that:

- all assignments in  $M$  are preserved in  $M'$ , and
- $M'$  is stable with respect to capacity  $c'$  where

$$c'(j) = \begin{cases} |M'^{-1}(h_j)| & \text{if } |M'^{-1}(h_j)| > c(j), \\ c(j) & \text{otherwise.} \end{cases} \quad (1)$$

Note that  $M'^{-1}(h_j)$  is the set of students assigned to  $h_j$  under  $M'$ . We say that  $M'$  is a *stable extension of  $M$  with respect to  $N$* .

The complete algorithm  $\text{STABLEEXTENSION}(M, c, N)$  is given in Figure 3 (Appendix). At a high level, the algorithm maintains a stable extension  $M_e$  of  $M$  with respect to a subset  $N'$  of  $N$ . At each step, a student  $s_i$  is added to  $N'$  and all possible assignments  $A$  of  $s_i$  that are compatible to  $M_e$  are identified. In other words, adding each assignment in  $A$  to  $M_e$  gives a stable extension of  $M$  with respect to  $N' \cup \{s_i\}$ . The algorithm branches to an assignment in  $A$  and continues to the next student. When  $N' = N$ , the current matching is returned. The algorithm then backtracks to a previous branching point and continues.

Figure 4 gives the subroutine for finding compatible assignments. Initially,  $A_i$  is set to be an empty set. The subroutine then goes through the preference list of  $s_i$  one by one in decreasing order. The considered school  $h$  is added to  $A_i$  and the subroutine terminates if at least one of the following happens:

- $h$  is  $\emptyset$ ,
- $h$  is under-filled,
- $h$  prefers  $s_i$  to  $\text{LPS-Assigned}(h)$  with respect to  $M_e$ .

FEASIBLEASSIGNMENT( $M_e, c, s_i$ ):

**Input:** Stable matching  $M_e$ , capacity  $c$ , student  $s_i$ .

**Output:** Set  $A_i$  of all possible assignments for  $s_i$ . Adding any assignment in  $A_i$  to  $M_e$  preserves stability.

1. Initialize  $A_i$  to the empty set.
2. For each  $h$  in  $l(s_i)$ , in decreasing order of preferences, do:
  - a. If  $h = \emptyset$  then Return  $A_i \cup \{\emptyset\}$ .
  - b. Else  $h = h_j$ :
    - i. If  $|M_e^{-1}(h_j)| < c(j)$  then Return  $A_i \cup \{h_j\}$ .
    - ii. If  $s_i$  appears before LPS-Assigned( $h_j$ ) then Return  $A_i \cup \{h_j\}$ .
    - iii. If  $s_i$  appears after LPS-Assigned( $h_j$ ) and before BS-Preferring( $h_j$ ) then  $A_i \leftarrow A_i \cup \{h_j\}$ .

■ **Figure 4** Algorithm for finding feasible matches of  $s_i$  w.r.t. current matching  $M_e$ .

Notice that in the last two scenarios above, if  $s_i$  was assigned to any school after  $h$  in her preference list,  $(s_i, h)$  would form a blocking pair. Assume none of the above scenarios happens. The subroutine adds  $h$  to  $A$  and continues if  $h$  prefers  $s_i$  to BS-Preferring( $h$ ). Otherwise,  $h$  prefers BS-Preferring( $h$ ) to  $s_i$ . Hence, assigning  $s_i$  to  $h$  would create a blocking pair. The subroutine continues to the next school in this case. The following lemma says that FEASIBLEASSIGNMENT correctly finds all possible assignments of a student, given the current matching, at each step.

► **Lemma 11.** *Let  $N'$  be the set of students assigned (possibly to  $\emptyset$ ) in  $M_e$ , i.e.,  $M_e$  is a stable extension of  $M$  with respect to  $N'$ . FEASIBLEASSIGNMENT( $M_e, c, s_i$ ) finds all possible assignments of  $s_i$  to  $H \cup \{\emptyset\}$  such that adding each assignment to  $M_e$  gives a stable extension of  $M$  with respect to  $N' \cup \{s_i\}$ .*

► **Lemma 12.** *FEASIBLEASSIGNMENT( $M_e, c, s_i$ ) returns at least one possible assignment.*

From Lemmas 11 and 12, we can prove the main theorem of this section:

► **Theorem 13.** *STABLEEXTENSION( $M, c, N$ ) enumerates all possible stable extension of  $M$  with respect to  $N$ . Moreover, the time between any two enumerations is  $O((k+n)m)$ .*

## 5 Mechanisms for Type B Settings

### 5.1 Setting B1

We first show some structural properties of minimum stable re-allocations in this setting.  $N' \subseteq N$ , defines the set of students who form blocking pairs with the current matching  $M$ .  $SM$  denote the set of stable matchings over the instance  $I = (S \cup N, H, c)$ , and  $MSR$  represents the set of all minimum stable re-allocations of  $M$ . For all  $s_i \in N$  we set  $M(s_i) = \emptyset$ .

► **Definition 14.**  $s_i \in S \cup N$  is **moved** in  $M' \in MSR$ , if  $M(s_i) \neq M'(s_i)$

► **Lemma 15.** *All minimum stable re-allocations of  $M$  move the same set of students,  $S_R$ .*

## 21:12 Stability-Preserving, Time-Efficient Mechanisms for School Choice in Two Rounds

Let  $H_R$ , be the set of schools that students in  $S_R$  are matched to in some minimum stable re-allocation then as an application of the Rural Hospitals Theorem we have:

► **Corollary 16.** *All students  $s_i \in S \cup N - S_R$  are matched to the same school in all minimum stable re-allocations. All minimum stable re-allocations will match students in  $S_R$  to schools in  $H_R$ . Moreover, if  $k$  students from  $S_R$  are matched to a school  $h_j \in H_R$ , then all minimum stable re-allocations will have  $k$  students from  $S_R$  matched to  $H_R$ .*

We denote the students in  $S \cup N - S_R$  as  $S_F$ , and let  $M_F$  represent the matching restricted to these students. Then for all  $M' \in MSR$  and  $s_i \in S_F$ ,  $M_F(s_i) = M(s_i) = M'(s_i)$ .

Consider the stable matching instance  $I'$ , defined below:

- (a)  $\forall s_i \in S_R$ ,  $\text{Barrier}(s_i) = \text{Best } h_j \in \text{Schools-FBPairs}(s_i)$  over all  $h_j \in H - H_R$
- (b)  $\forall h_j \in H_R$ ,  $\text{Barrier}(h_j) = \text{BS-Preferring}(h_j)$  among students in  $S_F$
- (c)  $\forall s_i \in S_R$ ,  $l'(s_i) = l(s_i)$ . Place the  $\emptyset$  to the immediate left of  $\text{Barrier}(s_i)$
- (d)  $\forall h_j \in H_R$   $l'(h_j) = l(h_j)$ . Place the  $\emptyset$  to the immediate left of  $\text{Barrier}(h_j)$
- (e) Let  $M'$  be some  $MSR$ , then  $c'(h_j) = |\{s_i \in S_R \mid M'(s_i) = h_j\}|$ 
  - $I' = (S_R, H_R, c')$  with preference lists  $l'(s_i), l'(h_j)$  defines a stable matching instance, with  $SM_{I'}$  denoting the set of all stable matchings over  $I'$ .

► **Lemma 17.**  $\forall M_{I'} \in SM_{I'}$ ,  $M' = M_{I'} \cup M_F$  is a minimum stable re-allocation. Moreover any  $M' \in MSR$  can be decomposed into  $M_{I'} \cup M_F$ , where  $M_{I'} \in SM_{I'}$ .

► **Lemma 18.**  $(MSR, \succeq)$  defines a sublattice of  $(SM, \succeq)$ .

ADDING NEW STUDENTS( $M, N$ ):

**Input:** Stable matching  $M$  and set  $N$ .

**Output:** Minimum stable re-allocation of  $M$ .

1.  $\forall s_i \in S_M : M'(s_i) \leftarrow M(s_i)$
2. While  $\exists s_i$  unmatched and  $(s_i, h_j)$  form a blocking pair do
  - a.  $h \leftarrow \text{Best possible } h_i$  in  $\text{Schools-FBPairs}(s_i)$
  - b. if  $h$  is filled to capacity then unmatched LPS-Assigned( $h$ )
  - c.  $M'(s_i) \leftarrow h$
3. Return  $M'$ .

■ **Figure 5** Mechanism  $\mathcal{M}_2$  for adding new students in round  $\mathcal{R}_2$ .

The proof that  $\mathcal{M}_2$  finds a MSR is provided in the Appendix. As a corollary of our proof we get:

► **Corollary 19.**  $\mathcal{M}_2$  produces a student-optimal minimum stable re-allocation.

► **Lemma 20.** *There exists a mechanism  $\mathcal{M}_3$ , that finds a school-optimal minimum stable re-allocation in polynomial time.*

## 5.2 Setting B2

A first approach to finding a minimum stable re-allocation in Setting B2 would be to run Gale-Shapley over the whole instance. However unlike Setting B1, Example 21 shows that this could require as many as  $|S|$  possible re-allocations.

► **Example 21.** Let there be  $n+1$  students and schools. The preference lists (mod  $n+1$ ) for any student  $s_i$  is  $(h_{i-1}, h_i, \dots, h_{i-2})$  and the preference list for any school  $h_j$  is  $(s_j, s_{j+1}, \dots, s_{j-1})$ . In round  $\mathcal{R}_1$ , all participants but  $h_{n+1}$  are present and each school has 1 seat. In round  $\mathcal{R}_2$ ,  $h_{n+1}$  arrives with capacity 1. The only stable matching from round  $\mathcal{R}_1$  would match each  $s_i$  to  $h_i$  and  $s_{n+1}$  would remain unmatched. Assigning  $s_{n+1}$  to  $h_{n+1}$  would result in a stable matching requiring no re-allocations. However, running Gale-Shapley over all participants would yield a matching of each  $s_i$  to  $h_{i-1}$ , but this matching requires  $n$  re-allocations.

► **Lemma 22.** *Each student weakly improves in any minimum stable reallocation.*

► **Remark 23.** The lattice structure shown in the previous section carries over to this Setting as well. This follows since both Settings B1 and B2 can be reduced to an instance where schools have unit capacity. Consider the unit capacity setting: a stable matching is found in round  $\mathcal{R}_1$ , and in round  $\mathcal{R}_2$  a set of new participants arrive on one side. Since each school has unit capacity, schools and students become interchangeable.

► **Lemma 24.**  *$(MSR, \succeq)$  is a sublattice of  $(SM, \succeq)$ . Moreover  $\mathcal{M}_2$  finds the school-optimal minimal stable re-allocation.*

ADDING NEW SCHOOLS( $M, H'$ ):

**Input:** Stable matching  $M$  and set  $H'$ .

**Output:** Minimum stable re-allocation of  $M$ .

1.  $\forall s_i \in S_M : M'(s_i) \leftarrow M(s_i)$
2. While  $\exists h_j \in H \cup H'$  with unmet-capacity and  $\text{BS-Preferring}(h_j) \neq \emptyset$ :
  - a. Break current match if exists of  $\text{BS-Preferring}(h_j)$
  - b.  $M' \leftarrow M' \cup (\text{BS-Preferring}(h_j), h_j)$
3. Return  $M'$ .

■ **Figure 6** Mechanism  $\mathcal{M}_2$  for adding new schools in round  $\mathcal{R}_2$ .

► **Lemma 25.** *There exists a mechanism  $\mathcal{M}_3$ , that finds a student-optimal minimum stable re-allocation in polynomial time.*

## 6 Incentive Compatibility

For the four settings discussed, it would be highly desirable if we could prove that mechanism  $\mathcal{M}_2$  in round  $\mathcal{R}_2$  is DSIC. We show that for Setting B1 that this truly is the case. Unfortunately for Settings A1, A2 and B2 we show that the current mechanisms outlined above are not incentive compatible. We relax DSIC and consider the weaker notion of a mechanism for which *incentive compatibility is a Nash equilibrium (ICNE)*. Under such a mechanism, a student cannot gain by misreporting her choices, if all other students are truthful. We show that no mechanisms in round  $\mathcal{R}_2$  for Setting A1, A2 and B2 can be even ICNE.

► **Lemma 26.**  $\mathcal{M}_2$  in Setting B1 is DSIC for students.

► **Lemma 27.** There is no pair of stability-preserving, ICNE mechanism  $(\mathcal{M}_1, \mathcal{M}_2)$  for Setting A1 and A2.

► **Lemma 28.** There is no pair of stability-preserving, ICNE mechanism  $(\mathcal{M}_1, \mathcal{M}_2)$  for Setting B2.

The key distinction for incentive compatibility between Setting A1,A2 and B1, is that in B1 if a student is unmatched after round  $\mathcal{R}_1$  it will remain unmatched after round  $\mathcal{R}_2$ . However in Settings A1, A2 we try to accommodate students who were unmatched after round  $\mathcal{R}_1$ , so they still have a chance to get matched in  $\mathcal{R}_2$ . This provides the possibility of affecting the matching produced in  $\mathcal{R}_1$  by misreporting their preference list so as to make the Barriers computed by  $\mathcal{M}_2$  more favorable for them.

## 7 NP-Hardness Results

► **Problem 29.** A different version of Setting A2, the City wants to extend original matching  $M$  so that it maximizes the number of students who get matched from  $L$ , and subject to this, minimize the number of students who get matched from  $N$ . ( $\max_L \min_N$ )

► **Problem 30.** Same setting as Problem 29, but the City wants to maximize the number of students who get matched from  $N$ , and subject to this, minimize the number of students who get matched from  $L$ . ( $\max_N \min_L$ )

► **Problem 31.** A set of new students  $N$  arrive in round  $\mathcal{R}_2$ . The City wants to extend the matching to include  $k$  students from  $N$ , such that it maximizes the number of students matched from  $L$ .

► **Problem 32.** In round  $\mathcal{R}_2$ , we are allowed to re-allocate some students matched in round  $\mathcal{R}_1$  in order to match more students from  $L$ . Find a stable matching that maximizes the number of students matched from  $L$ , and subject to this, minimizes the number of re-allocations made.

► **Problem 33.** In the single round setting, given a set of students, and schools with strictly ordered preference lists  $l(s), l(h)$  respectively, and a weight function  $w(j)$  over the edges of students to schools, find a vector of capacities for the schools and a stable matching with respect to this vector that maximizes the total weight.

► **Theorem 34.** Problems 29, 30, 31, 32, and 33 are NP-hard.

---

### References

- 1 Atila Abdulkadiroglu, Yeon-Koo Che, Parag A Pathak, Alvin E Roth, and Olivier Tercieux. Minimizing justified envy in school choice: The design of new orleans' oneapp. Technical report, National Bureau of Economic Research, 2017.
- 2 Atila Abdulkadiroglu, Parag A Pathak, and Alvin E Roth. Strategy-proofness versus efficiency in matching with indifferences: Redesigning the NYC high school match. *American Economic Review*, 99(5):1954–78, 2009.
- 3 Atila Abdulkadiroglu and Tayfun Sonmez. School choice: A mechanism design approach. *American economic review*, 93(3):729–747, 2003.
- 4 Atila Abdulkadiroglu and Tayfun Sonmez. Matching markets: Theory and practice. *Advances in Economics and Econometrics*, 1:3–47, 2013.
- 5 Tommy Andersson, Umut Dur, Sinan Ertemel, Onur Kesten, et al. Sequential school choice with public and private schools. Technical report, Lund University, 2018.

- 6 Battal Dogan and M Bumin Yenmez. When does an additional stage improve welfare in centralized assignment?, 2018.
- 7 Laura Doval. A theory of stability in dynamic matching markets. Technical report, Technical report, mimeo, 2018.
- 8 Lester E Dubins and David A Freedman. Machiavelli and the gale-shapley algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981.
- 9 Lester E Dubins and David A Freedman. Machiavelli and the Gale-Shapley algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981.
- 10 Umut Dur and Onur Kesten. Sequential versus simultaneous assignment systems and two applications. *Economic Theory*, pages 1–33, 2014.
- 11 Federico Echenique and Juan Sebastián Pereyra. Strategic complementarities and unraveling in matching markets. *Theoretical Economics*, 11(1):1–39, 2016.
- 12 Itai Feigenbaum, Yash Kanoria, Irene Lo, and Jay Sethuraman. Dynamic matching in school choice: Efficient seat reallocation after late cancellations, 2018.
- 13 Simons Institute for the Theory of Computing. Online and matching-based market design, 2019. URL: <https://simons.berkeley.edu/programs/market2019>.
- 14 David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- 15 Joseph Gasper, Stefanie DeLuca, and Angela Estacion. Switching schools: Revisiting the relationship between school mobility and high school dropout. *American Educational Research Journal*, 49(3):487–519, 2012.
- 16 Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.
- 17 Sangram V Kadam and Maciej H Kotowski. Multiperiod matching. *International Economic Review*, 59(4):1927–1947, 2018.
- 18 Parag A Pathak. The mechanism design approach to student assignment. *Annu. Rev. Econ.*, 3(1):513–536, 2011.
- 19 Alvin E. Roth. Stability and polarization of interests in job matching. *Econometrica: Journal of the Econometric Society*, pages 47–57, 1984.
- 20 Alvin E Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica: Journal of the Econometric Society*, pages 425–427, 1986.
- 21 Alvin E Roth. What have we learned from market design? *Innovations: Technology, Governance, Globalization*, 3(1):119–147, 2008.
- 22 Alvin E. Roth. Al Roth’s game theory, experimental economics, and market design page, 2016. URL: <http://stanford.edu/~alroth/alroth.html#MarketDesign>.
- 23 Alvin E. Roth and Lloyd S. Shapley. Nobel Memorial Prize in Economics, 2012. URL: <https://www.nobelprize.org/prizes/economic-sciences/2012/summary/>.
- 24 Alexander Westkamp. An analysis of the german university admissions system. *Economic Theory*, 53(3):561–589, 2013.



# New Verification Schemes for Frequency-Based Functions on Data Streams

Prantar Ghosh

Dartmouth College, Hanover, NH, USA

prantar.ghosh.gr@dartmouth.edu

---

## Abstract

We study the general problem of computing *frequency-based functions*, i.e., the sum of any given function of data stream frequencies. Special cases include fundamental data stream problems such as computing the number of distinct elements ( $F_0$ ), frequency moments ( $F_k$ ), and heavy-hitters. It can also be applied to calculate the maximum frequency of an element ( $F_\infty$ ).

Given that exact computation of most of these special cases provably do not admit any sublinear space algorithm, a natural approach is to consider them in an enhanced data streaming model, where we have a computationally unbounded but untrusted *prover* that can send *proofs* or help messages to ease the computation. Think of a memory-restricted client delegating the computation to a powerful cloud service. The client does not blindly trust the cloud, and with its limited memory, it wants to *verify* the proof that the cloud sends. Chakrabarti et al. (ICALP '09) introduced this model as the *annotated data streaming model* and showed that multiple problems including exact computation of frequency-based functions – that have no sublinear algorithms in basic streaming – do have algorithms, also called *schemes*, in the annotated streaming model with both space and proof-length sublinear in the input size.

We give a general scheme for computing any frequency-based function with both space usage and proof-size of  $O(n^{2/3} \log n)$  bits, where  $n$  is the size of the universe. This improves upon the best known bound of  $O(n^{2/3} \log^{4/3} n)$  given by the seminal paper of Chakrabarti et al. and as a result, also improves upon the best known bounds for the important special cases of computing  $F_0$  and  $F_\infty$ . We emphasize that while being quantitatively better, our scheme is also qualitatively better in the sense that it is simpler than the previously best scheme that uses intricate data structures and elaborate subroutines. Our scheme uses a simple technique tailored for this model: the verifier solves the problem partially by running an algorithm known to be helpful for it in the basic (sans prover) streaming model and then takes the prover's help to solve the remaining part.

**2012 ACM Subject Classification** Theory of computation → Streaming models; Theory of computation → Interactive proof systems; Computer systems organization → Cloud computing

**Keywords and phrases** data streams, interactive proofs, Arthur-Merlin

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.22

**Funding** *Prantar Ghosh*: Work supported in part by NSF under award CCF-1907738

**Acknowledgements** The author would like to thank Amit Chakrabarti and Justin Thaler for several helpful discussions. He is also grateful to the anonymous FSTTCS 2020 reviewers for their valuable comments, especially to the reviewer who gave a sketch of a scheme using a randomized estimation algorithm, pointing out that it can handle longer streams and that determinism isn't strictly necessary for the subroutine.

## 1 Introduction

Interactive proof systems have contributed a very important conceptual message to computer science: it is possible for a computationally bounded entity to reduce its computational cost for a problem if it is only required to verify a proof of the solution instead of finding a solution on its own. This concept led to celebrated results such as  $IP = PSPACE$  [28] and the PCP Theorems [3, 4]. It is natural to incorporate this idea to deal with challenging problems in



© Prantar Ghosh;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 22; pp. 22:1–22:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

massive data streams so as to reduce the impractical computational costs for such problems. This incorporation led to the following setting: a space-restricted client reading a huge data stream outsources the computation to a more powerful entity, such as a cloud service, with unbounded space. The cloud sends the result of the computation to the client who refuses to blindly trust it since it might be malicious or might have incurred some hardware failure. Therefore, the cloud (henceforth named “Prover”) also sends the client (henceforth named “Verifier”) a *proof* in support of its results. Verifier needs to use his limited space to collect sufficient information from the stream so as to verify the proof. In the case that Prover is honest, Verifier can use it as a help message to find the solution to the underlying problem. Otherwise, he rejects the proof. This combination of data streaming with prover-verifier systems has been fruitful: multiple works [1, 6, 7, 8, 9, 10, 12, 14, 21, 22, 30] have shown that several problems that are provably intractable in the basic data streaming model turn out to be solvable in prover-enhanced models using verification space and proof-length sublinear in the input size.

Chakrabarti et al. [7] formally defined this enhanced data streaming model as the *annotated data streaming model*. An algorithm in this model is called a *scheme*. In designing a scheme, the two important complexity parameters that we need to focus on are the *space* used by Verifier and the *size of the proof* sent by Prover. A scheme that has a proof-length of  $O(h)$  bits and uses  $O(v)$  bits of space is called an  $(h, v)$ -*scheme*.

Since its inception, data streaming algorithms have been extensively studied for fundamental statistical problems such as counting the number of distinct elements in a stream ( $F_0$ ) [2, 5, 15, 20], the  $k$ th frequency moment for  $k > 0$  ( $F_k$ ) [2, 16, 18, 32], the maximum frequency of an element ( $F_\infty$ ) [2, 19], and the  $\ell_p$ -norm of the frequency vector for some  $p \geq 0$  [19, 20, 26]. All of these problems are special cases of (or can be solved by easily applying) the general problem of computing *frequency-based functions*: given a function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ , find  $\sum_{j=1}^n g(f_j)$ , where, for each  $j$  in the universe  $\{1, \dots, n\}$ ,  $f_j$  is the frequency of the  $j$ th element. This general problem was notably addressed by the celebrated seminal paper by Alon, Matias, and Szegedy [2]: they asked for a characterization of precisely which frequency-based functions can be approximated efficiently in the basic streaming model. The aforementioned paper by Chakrabarti et al. [7] studied such statistical problems in the annotated streaming setting and gave several interesting schemes. In particular, for the general problem of computing frequency-based functions, they gave an  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -scheme. Their scheme uses an intricate data structure with binary trees and calls upon a subroutine for heavy-hitters that uses an elaborate framework called *hierarchical heavy hitters*.

Given how general the problem is, with several important special cases having numerous applications, it is important and beneficial to have a *simple* scheme for the general problem. In this work, we design such a simple scheme that uses the most basic and classical data structure for frequency estimation: the Misra-Gries summary [25]. Our scheme ends up improving the best known complexity bounds for the problem: we give an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme. No better bounds or simpler algorithms were known even for the special cases of computing  $F_0$  or  $F_\infty$ . Our result thus simplifies and improves the bounds for these problems as well.

The aforementioned scheme works for streams of length  $m = O(n)$ , an assumption that was also made by Chakrabarti et al. [7]. However, their scheme can be made to work for longer turnstile streams as long as  $\|\mathbf{f}\|_1 = O(n)$ . We show how to use the Count-Median Sketch [13], an estimation algorithm with stronger guarantees than Misra-Gries, to get a scheme with similar complexity bounds for these long streams. But since the Count-Median Sketch is randomized (contrary to Misra-Gries), we incur a non-zero completeness error for this scheme. The high-level idea of both our schemes is the following: we use the estimation algorithm as a primitive to “partially” solve the problem. Prover then helps Verifier with the “remaining” unsolved part.

## 1.1 Setup and Terminology

We formalize the setting described above. A *scheme* for computing a function  $g(\sigma)$  of the input stream  $\sigma$  is a triple  $(\mathcal{H}, \mathcal{A}, \text{out})$ , where  $\mathcal{H}$  is a function that Prover uses to generate the help message or proof-stream for  $\sigma$ , given by  $\mathcal{H}(\sigma)$ ,  $\mathcal{A}$  is a data streaming algorithm that Verifier runs on the stream  $\sigma$  using a random string  $R$  to produce a summary  $\mathcal{A}_R(\sigma)$ , and  $\text{out}$  is a streaming algorithm that Verifier runs on the proof-stream  $\mathcal{H}(\sigma)$  and also uses  $\mathcal{A}_R(\sigma)$  and  $R$  to generate an output  $\text{out}_R(\mathcal{H}(\sigma), \mathcal{A}_R(\sigma))$  in  $\text{range}(g) \cup \perp$ , where the symbol  $\perp$  denotes rejection of the proof. Note that if the proof-length  $|\mathcal{H}(\sigma)|$  is larger than the memory of Verifier, then he needs to process  $\mathcal{H}(\sigma)$  as a stream.

A scheme  $(\mathcal{H}, \mathcal{A}, \text{out})$  has completeness error  $\varepsilon_c$  and soundness error  $\varepsilon_s$  if it satisfies

- (completeness)  $\forall \sigma : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), \mathcal{H}(\sigma)) = g(\sigma)] \geq 1 - \varepsilon_c$ ;
- (soundness)  $\forall \sigma, H : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), H) \notin \{g(\sigma), \perp\}] \leq \varepsilon_s$ .

Informally, this means that an honest Prover can convince Verifier to produce the correct output with high probability. Again, if Prover is dishonest, then, with high probability, Verifier rejects the proof. We usually aim for  $\varepsilon_c, \varepsilon_s \leq 1/3$  (they can be boosted down using standard techniques incurring a small increase in the space usage). A scheme is said to have *perfect completeness* if  $\varepsilon_c = 0$ .

The *hcost* (short for “help cost”) of a scheme  $(\mathcal{H}, \mathcal{A}, \text{out})$  is defined as  $\max_{\sigma} |\mathcal{H}(\sigma)|$ , i.e., the maximum number of bits required to express a proof. The *vcost* (short for “verification cost”) is the maximum bits of space used by the algorithms  $\mathcal{A}_R(\sigma)$  and  $\text{out}_R(\sigma)$ , where the maximum is taken over all inputs  $\sigma$  and possible random strings  $R$ . A scheme with  $\text{hcost } O(h)$  and  $\text{vcost } O(v)$  is called an  $(h, v)$ -scheme. An  $(h, v)$ -scheme is interesting if  $h > 0$  and  $v$  is asymptotically smaller than the best bound achievable for  $h = 0$ , i.e., in the basic (sans prover) streaming model.

## 1.2 Our Results and Techniques

In this section, we state our results and give an overview of our techniques.

**Results.** Given a stream with elements in  $[n]$ , let  $\mathbf{f}$  denote its frequency vector  $\langle f_1, f_2, \dots, f_n \rangle$ , where  $f_j$  is the frequency of the  $j$ th element. A *frequency-based function* is a function  $G(\mathbf{f})$  of the form  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  for some function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ .

Our main result is captured in the following theorem which we prove in Section 3.2.1.

► **Theorem 1.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream of length  $m = O(n)$ . The scheme is perfectly complete and has soundness error at most  $1/\text{poly}(n)$ .*

With some modifications, we obtain a similar scheme for longer streams at the cost of imperfect completeness. This is given by the following theorem which we prove in Section 3.2.2.

► **Theorem 2.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ . The scheme has completeness and soundness errors at most  $1/3$ .*

As a consequence, we get schemes with the same complexity bounds for the problems of computing  $F_0$ ,  $F_\infty$ , and checking multiset inclusion (see Corollary 5 for formal definition). Just as for frequency-based functions, our schemes also improve upon the best known bounds for these special cases and applications<sup>1</sup>. We discuss these results in detail in Section 3.3.

<sup>1</sup> Computing  $F_k$  for constant  $k > 0$  is a well-studied special case for which better bounds are known [7].

► **Corollary 3.** For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_0$ , the number of distinct elements with non-zero frequency, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .

► **Corollary 4.** For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_\infty$ , the maximum frequency of an element, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .

► **Corollary 5.** Let  $X, Y \subseteq [n]$  be multisets of size  $O(n)$ . Given a stream where elements of  $X$  and  $Y$  arrive in interleaved manner, there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for determining whether  $X \subseteq Y$ .

**Techniques.** Computing frequency-based functions is challenging simply because we don't have enough space to store all the exact frequencies. However, there are efficient small-space algorithms – e.g., Misra-Gries algorithm [25], Count-Median Sketch [13] – that return reasonably good *estimates* of the frequencies. We use such an algorithm as a primitive in our schemes. The estimates returned partially solve the problem by helping us identify the “heavy-hitters” or the most frequent items. There cannot be too many heavy-hitters and hence, the all-powerful Prover can send Verifier the exact frequencies of these elements (which of course need to be verified) without too much communication. On the other hand, the rest of the elements, though large in number, have relatively small frequency. We show a way to encode the answer in terms of a low-degree polynomial when the frequencies are small. Prover can then send us this polynomial using few bits, enabling us to solve the problem with small communication overall.

We remark that the high-level technique used in our first scheme – using Misra-Gries as a subroutine – might be more widely applicable than that used in the second one, i.e., using Count-Median Sketch. This is because Misra-Gries is deterministic while Count-Median is randomized. In general, both Prover and Verifier can locally run a *deterministic* algorithm on the input, and then, Prover can send messages based on the final state of that algorithm. Note that it isn't clear if a *randomized* algorithm can always help in this regard since we assume that Prover and Verifier do not have access to shared randomness<sup>2</sup>. Hence, the final states of the algorithm might vary drastically for Prover and Verifier if they run it locally with their own private randomness. For our problem, we don't run into this issue since we don't require Prover to know the exact output of the Verifier's local estimation algorithm.

Other techniques used are pretty standard in this area. We use techniques based on the famous *sum-check protocol* of Lund et al. [24] that encodes answers as sum of low-degree polynomials. In our case, where Prover sends only a single message to Verifier, a quantity of interest is expressed as the sum of evaluations of a low-degree univariate polynomial. Since the polynomial has low-degree, it can be expressed with a small number of monomials. Thus, Prover needs only a few bits to express the set of coefficients that describe the polynomial, leading to short proof-length. Moreover, to verify the authenticity of the polynomial, Verifier needs to evaluate it at just a single random point, the space for which he can afford. The main challenge in this technique is to find the proper low-degree polynomials to encode the answer,

---

<sup>2</sup> This assumption is made so that it corresponds to the MA communication model. Access to shared randomness corresponds to the AMA communication model where better bounds are known [17].

and in this work, we give such new polynomial encodings for the underlying sub-problems. Another standard technique we use is the *shaping technique* that transforms a one-dimensional vector into a two-dimensional array. On a high level, this helps in “distributing” the work between Prover and Verifier as they each “take care of” a single dimension. Pertaining to the streaming model, we exploit the popular technique of *linear sketching* where we express a quantity of interest as a linear combination of the stream updates, which helps us to maintain the quantity dynamically as the stream arrives.

### 1.3 Related Work

Early works on the concept of stream outsourcing and verification were done by the database community [23, 27, 31, 33]. Motivated by these works, Chakrabarti et al. [7] abstracted out and formalised the theoretical aspects of the settings. They defined two types of stream verification settings: (i) the *annotated data streaming* setting – calling the schemes as *online schemes* – where Prover and Verifier read the input stream together and Prover sends help messages during and/or after the stream arrival based on the part of the stream she has seen so far, and (ii) the *prescient* setting where Prover knows the entire stream upfront, i.e., before Verifier sees it, and can send help messages accordingly. Several subsequent works [6, 9, 10, 12, 21, 30] studied these non-interactive models. Natural generalizations of the model, where we allow multiple rounds of interaction between Prover and Verifier, have also been explored. These include *Arthur-Merlin streaming protocols* (Prover is named “Merlin” and Verifier is named “Arthur” following a long-standing tradition in complexity theory) of Gur and Raz [17] and the *streaming interactive proofs* (SIP) of Cormode et al. [14]. The latter setting was further studied by multiple works [1, 8, 22]. We refer the reader to the expository article by Thaler [29] for a detailed survey of this area.

We state the results with the standard assumption [7, 14] that  $m = O(n)$ . Chakrabarti et al. [7] gave two schemes for computing any general frequency-based function: an online  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -scheme and a prescient  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme. They noted that the schemes apply to get best known schemes for the special cases of computing the number of distinct elements ( $F_0$ ), the maximum frequency ( $F_\infty$ ), and for checking multiset inclusion. They also showed a lower bound that any online or prescient  $(h, v)$ -scheme for the problem (even for the aforementioned special cases) requires  $hv \geq n$ . They designed schemes with better bounds for certain other frequency-based functions, often matching this lower bound up to polylogarithmic factors. For instance, for any  $hv = n$ , they gave an online  $(k^2 h \log n, kv \log n)$ -scheme for calculating the  $k$ th frequency moment  $F_k$  for any positive integer  $k$ , and a  $(\phi^{-1} \log^2 n + h \log n, v \log n)$ -scheme for computing the  $\phi$ -heavy hitters (elements with frequency of at least a  $\phi$ -fraction of the stream length).

The specific problem of computing  $F_0$  has been studied by multiple works in various stream verification models. Cormode, Mitzenmacher, and Thaler [11] studied the problem in the stronger SIP-model and gave a  $(\log^3 n, \log^2 n)$ -SIP with  $O(\log^2 n)$  rounds of communication. For the case where we restrict the number of rounds to  $O(\log n)$ , Cormode, Thaler, and Yi [14] gave a  $(\sqrt{n} \log^2 n, \log^2 n)$ -SIP. Klauck and Prakash [22] improved this to a  $(\log^4 n \log \log n, \log^2 n \log \log n)$ -SIP. Gur and Raz [17] designed an  $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$ -AMA-streaming protocol<sup>3</sup> (the  $\tilde{O}(\cdot)$  notation hides polylog( $n$ ) factors) for  $F_0$ .

<sup>3</sup> AMA stands for the communication pattern Arthur-Merlin-Arthur

## 2 Preliminaries

Here, we discuss the streaming models we study and some standard results that we use in our schemes.

Throughout this paper, the stream elements come from the universe  $[n] := \{1, \dots, n\}$  and the stream length is  $m$ . In the *turnstile* streaming model, tokens are of the form  $(j, \Delta) \in [n] \times \mathbb{Z}$ , which means  $\Delta$  copies of the element  $j$  are inserted (resp. deleted) if  $\Delta > 0$  (resp.  $\Delta < 0$ ). The *cash register* or *insert-only* streaming model is the special case when  $\Delta$  is always positive. In this paper, for simplicity, we assume unit updates, i.e.,  $\Delta \in \{-1, 1\}$  always. The assumption can be easily removed by looking at an update as a collection of multiple unit updates.

For a stream  $\sigma = \langle (a_1, \Delta_1), \dots, (a_m, \Delta_m) \rangle$ , the *frequency vector*  $\mathbf{f}(\sigma)$  is defined as  $\langle f_1, \dots, f_n \rangle$  where  $f_j$  is the *frequency* of element  $j$ , given by  $f_j := \sum_{i \in [m]: a_i=j} \Delta_i$ . We denote *estimates* of  $f_j$  by  $\hat{f}_j$ . We drop the argument  $\sigma$  from  $\mathbf{f}(\sigma)$  when the stream is clear from the context.

In our schemes, we use the standard technique of *sketching* a frequency vector by evaluating its *low-degree extension* at a random point. We explain what this means. We transform (or *shape*) our frequency vector of length  $n$  into a 2-dimensional  $d_1 \times d_2$  array  $f$ , where  $d_1 d_2 = n$ , using some canonical bijection from  $[n]$  to  $[d_1] \times [d_2]$ . This means that the domain of the function  $f$  can now be seen as  $[d_1] \times [d_2]$ . We work on a finite field  $\mathbb{F}$  with large enough characteristic such that the values don't "wrap around" under operations in  $\mathbb{F}$ . By Lagrange's interpolation, there is a unique polynomial  $\tilde{f}(X, Y) \in \mathbb{F}[X, Y]$  with  $\deg_X(\tilde{f}) = d_1 - 1$  and  $\deg_Y(\tilde{f}) = d_2 - 1$  such that  $\tilde{f}(x, y) = f(x, y)$  for all  $(x, y) \in [d_1] \times [d_2]$ . We call  $\tilde{f}$  the *low-degree  $\mathbb{F}$ -extension* of  $f$ . For each  $(x, y) \in [d_1] \times [d_2]$ , we have "Lagrange basis polynomials" defined as

$$\delta_{x,y}(X, Y) := \left( \prod_{x_i \in [d_1] \setminus \{x\}} \frac{X - x_i}{x - x_i} \right) \cdot \left( \prod_{y_i \in [d_2] \setminus \{y\}} \frac{Y - y_i}{y - y_i} \right) \quad (1)$$

We can write  $\tilde{f}$  as a linear combination of these polynomials as follows:

$$\tilde{f}(X, Y) = \sum_{(x,y) \in [d_1] \times [d_2]} f(x, y) \delta_{x,y}(X, Y)$$

In particular, if  $f$  is built up from a stream of turnstile updates  $\langle (x, y)_j, \Delta_j \rangle$ , then

$$\tilde{f}(X, Y) = \sum_j \Delta_j \delta_{(x,y)_j}(X, Y). \quad (2)$$

Thus, we can use Equation (2) to *maintain*  $\tilde{f}(x^*, y^*)$  at some fixed point  $(x^*, y^*)$  dynamically with stream updates. We formalize this in the following fact.

► **Fact 6.** *Given a point  $(x^*, y^*) \in \mathbb{F}^2$  and a stream of updates to an initially-zero  $d_1 \times d_2$ -dimensional array  $f$ , we can maintain  $\tilde{f}(x^*, y^*)$  using  $O(\log |\mathbb{F}|)$  space. For implementation details and generalizations, see Cormode et al. [14].*

An important subroutine in one of our schemes is the classic Misra-Gries algorithm for frequency estimation [25] which, given an input stream of  $m$  elements and a fraction  $\phi$ , estimates the frequency of the stream elements within an additive factor of  $\phi m$ . We recall this algorithm in Algorithm 1.

Informally, the algorithm does the following: it keeps an array or “dictionary”  $K$  indexed by “keys” that are elements of the stream and each of them has an associated counter  $K[i]$ . At any point of time, the array has at most  $\lceil \phi^{-1} \rceil$  keys. When a stream element arrives, it increments the counter for the element if it’s present in the keys (it includes it in the keys if there are less than  $\lceil \phi^{-1} \rceil$  keys), and otherwise decrements the counter of every key. If a counter for a key becomes 0, it is removed from  $K$ . Finally, the estimate  $\hat{f}_j$  is given by  $K[j]$  (which is 0 if  $j$  is not in the keys). The guarantees of the algorithm is given in Fact 7.

■ **Algorithm 1** [25] Misra-Gries algorithm for frequency estimates in insert-only streams.

---

**Input:** Stream  $\sigma$ ;  $\phi \leq 1$

1: Initialize  $K \leftarrow$  empty array

**Process**(token  $j \in \sigma$ ):

2: **if**  $j \in \text{keys}(K)$  **then**

3:      $K[j] \leftarrow K[j] + 1$

4: **else**

5:     **if**  $|\text{keys}(K)| < \lceil \phi^{-1} \rceil$  **then**

6:          $K[j] \leftarrow 1$

7:     **else**

8:         **for**  $i \in \text{keys}(K)$  **do**:

9:              $K[i] \leftarrow K[i] - 1$

10:         **if**  $K[i] = 0$  **then** remove  $i$  from  $\text{keys}(K)$

**Output:**

11: **for**  $j \in [n]$  **do**:

12:     **if**  $j \in \text{keys}(K)$  **then** **return**  $\hat{f}_j = K[j]$ ; **else** **return**  $\hat{f}_j = 0$

---

► **Fact 7** ([25]). *For an insert-only stream of  $m$  elements in  $[n]$ , given any  $\phi \leq 1$ , Algorithm 1 uses  $O(\phi^{-1}(\log n + \log m))$  space and returns frequency estimates  $\langle \hat{f}_j : j \in [n] \rangle$  such that, for all tokens  $j \in [n]$ , we have  $f_j - \phi m \leq \hat{f}_j \leq f_j$ .*

Note that this algorithm was designed for insert-only streams and doesn’t work for turnstile streams. To use it for turnstile streams, we need to make appropriate modifications (which we do in Section 3.1).

### 3 Computing Frequency-based Functions in Turnstile Streams

Let  $\mathbf{f}$  be the frequency vector of a stream as defined in Section 2. Recall that a *frequency-based function* is a function  $G(\mathbf{f})$  of the form  $G(\mathbf{f}) := \sum_{j \in [n]} g(f_j)$  for some function  $g : \mathbb{Z} \rightarrow \mathbb{Z}^+$ . In this section, we obtain an improved  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function for some predetermined function  $g$ . As stated earlier, we design a scheme exploiting the Misra-Gries algorithm (Algorithm 1). We want to use it as a subroutine in our problem for turnstile streams, but it works only in the insert-only model. Therefore, in Section 3.1, we provide a simple extension of the algorithm that attains a similar guarantee for turnstile streams. In Section 3.2, first, we use this extended Misra-Gries (EMG) algorithm as a subroutine for our scheme for computing frequency-based functions. Next, we show that we can instead use the Count-Median Sketch [13] to make our scheme work for longer streams. In Section 3.3, we discuss some important applications of our schemes.

### 3.1 Extension of Misra-Gries Algorithm for Turnstile Streams

The extended Misra-Gries algorithm (henceforth called “EMG algorithm”) works as follows: we process the positive and negative updates separately in two parallel copies of Algorithm 1 to estimate the total positive update and (absolute value of) the total negative update. In the second copy, we can actually think of the updates as “increments” since only negative updates are processed there. Thus, what we are actually estimating is the absolute value of the total negative update.

For each  $j$ , let the total positive update be  $f_j^+$  and (absolute value of) the total negative update  $f_j^-$ . Then, the actual frequency is  $f_j = f_j^+ - f_j^-$ . Denote the corresponding estimates given by the copies of Algorithm 1 by  $\hat{f}_j^+$  and  $\hat{f}_j^-$ . Then  $\hat{f}_j := \hat{f}_j^+ - \hat{f}_j^-$  gives a similar guarantee as Fact 7 for turnstile streams; this time, we also incur an additive error of  $\phi m$  on the upper bound.

To see this, note that by Fact 7, we have,  $\forall j \in [n]$ ,

$$f_j^+ - \phi m \leq \hat{f}_j^+ \leq f_j^+ \quad (3)$$

$$f_j^- - \phi m \leq \hat{f}_j^- \leq f_j^- \quad (4)$$

Thus, Equations (3) and (4) give  $f_j^+ - f_j^- - \phi m \leq \hat{f}_j^+ - \hat{f}_j^- \leq f_j^+ - f_j^- + \phi m$ , i.e.,

$$f_j - \phi m \leq \hat{f}_j \leq f_j + \phi m \quad (5)$$

Hence, this time we get double sided error. This estimate would suffice for getting our desired scheme. Therefore, we get the following lemma.

► **Lemma 8.** *Given a turnstile stream of  $m$  elements in  $[n]$ , the EMG algorithm uses  $O(\phi^{-1}(\log n + \log m))$  space and returns a summary  $\langle \hat{f}_j : j \in [n] \rangle$  such that, for all  $j \in [n]$ , we have  $f_j - \phi m \leq \hat{f}_j \leq f_j + \phi m$ .*

► **Remark 9.** The guarantee given by the EMG algorithm may not be very useful in general for turnstile streams. This is because the total number of stream updates  $m$  can be huge, whereas the frequency of each token can be small since we allow both increments and decrements in the turnstile model. The classic Misra-Gries algorithm for insert-only model, on the other hand, has a good guarantee (Fact 7) since  $m = \|\mathbf{f}\|_1$  in this model. However, for our purpose, the guarantee in Lemma 8 is good enough since we assume that  $m = O(n)$ .

### 3.2 Schemes for Frequency-based Functions

First, in Section 3.2.1, we describe a protocol for computing frequency-based functions in turnstile streams of length  $O(n)$  and prove Theorem 1. Next, in Section 3.2.2, we show that the scheme can be modified to work for any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , proving Theorem 2. The completeness error in the latter scheme is, however, non-zero.

#### 3.2.1 Perfectly Complete Scheme for $O(n)$ -Length Streams

As in prior works [7, 14], we solve the problem for stream length  $m = O(n)$ . Hence, by Lemma 8, the EMG algorithm takes  $O(\phi^{-1} \log n)$  space and gives, for some constant  $c$ ,

$$\forall j \in [n] : f_j - \phi cn \leq \hat{f}_j \leq f_j + \phi cn. \quad (6)$$

Set  $\phi = (cn^{2/3})^{-1}$ . Therefore, we have an  $O(n^{2/3} \log n)$  space algorithm that guarantees

$$\forall j \in [n] : f_j - n^{1/3} \leq \hat{f}_j \leq f_j + n^{1/3}.$$

Let  $K$  denote the set of keys in the final state of the EMG algorithm for the setting of  $\phi = 1/(cn^{2/3})$ . Observe that if  $\hat{f}_j = 0$  for some  $j$  (i.e.,  $j \notin K$ ), we know that  $f_j \in [-n^{1/3}, n^{1/3}]$ .

Define  $h(j) = \mathbb{I}\{j \notin K\}$  where  $\mathbb{I}$  is the 0-1 indicator function. We have

$$\sum_{j \in [n]} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \notin K} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \in [n]} g(f(j))h(j)$$

Let  $L := \sum_{j \in K} g(f(j))$  and  $R := \sum_{j \in [n]} g(f(j))h(j)$ . We shall compute  $L$  and  $R$  separately and add them to get the desired answer.

We *shape* (see Section 2) the 1D array  $[n]$  into a 2D  $n^{1/3} \times n^{2/3}$  array. Thus, we get

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} g(f(x, y))h(x, y)$$

As is standard [7], we assume that the range of the function  $g$  is upper bounded by some polynomial in  $n$ , say  $n^p$ . Pick a prime  $q$  such that  $n^{p+1} < q < 2n^{p+1}$ . We will work in the finite field  $\mathbb{F}_q$  and the upper bound on the range of  $g$  ensures that  $G(\mathbf{f})$  will not “wrap around” under arithmetic in  $\mathbb{F}_q$ .

Let  $\tilde{f}, \tilde{h}$  be polynomials of lowest degree over the finite field  $\mathbb{F}_q$  that agree with  $f, h$  respectively at all values in  $[n^{1/3}] \times [n^{2/3}]$ . Note that, by Lagrange’s interpolation, both  $\tilde{f}$  and  $\tilde{h}$  have degrees  $n^{1/3} - 1$  and  $n^{2/3} - 1$  in the two variables (see Section 2). Again, let  $\tilde{g}$  denote the polynomial of lowest degree that agrees with  $g$  at all values in  $[-n^{1/3}, n^{1/3}]$ . Thus,  $\tilde{g}$  has degree  $2n^{1/3}$ .

Therefore, we have

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(x, y))\tilde{h}(x, y)$$

i.e., we can write

$$R = \sum_{x \in [n^{1/3}]} P(x), \tag{7}$$

where the polynomial  $P$  is given by

$$P(X) = \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(X, y))\tilde{h}(X, y) \tag{8}$$

To compute  $L$ , it suffices to obtain the values  $f_j$  for all  $j \in K$  since  $g$  is predetermined. In our protocol, Prover would send values  $f'_j$  that she claims to be  $f_j$  for all  $j \in K$ . Define

$$T := \sum_{j \in K} (f_j - f'_j)^2$$

Note that we have  $f_j = f'_j$  for each  $j$  if and only if  $T = 0$ . Set  $f'_j := 0$  for all  $j \notin K$ . Thus, we can rewrite  $T$  as

$$T = \sum_{j \in [n]} (f_j - f'_j)^2 (1 - h(j))$$

Using shaping as before, we get

$$T = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} (f(x, y) - f'(x, y))^2 (1 - h(x, y)) \tag{9}$$

Let  $\tilde{f}'$  denote the polynomial of lowest degree over  $\mathbb{F}_q$  that agrees with  $f'$  at all values in  $[n^{1/3}] \times [n^{2/3}]$ . Therefore, we have

$$T = \sum_{x \in [n^{1/3}]} Q(x) \quad (10)$$

where the polynomial  $Q$  is given by

$$Q(X) = \sum_{y \in [n^{2/3}]} (\tilde{f}(X, y) - \tilde{f}'(X, y))^2 (1 - \tilde{h}(X, y)). \quad (11)$$

We are now ready to describe the protocol.

**Stream processing.** Verifier picks  $r \in \mathbb{F}_q$  uniformly at random. As the stream arrives, he maintains  $\tilde{f}(r, y)$  for all  $y \in [n^{2/3}]$  (Fact 6). In parallel, he runs the EMG algorithm setting  $\phi = (cn^{2/3})^{-1}$ .

**Help message.** Prover sends polynomials  $P'$  and  $Q'$ , and values  $f'_j$  for all  $j \in K$ . She claims that  $P', Q', f'$  are identical to  $P, Q, f$  respectively. The polynomials are sent as streams of their coefficients following some canonical order of their monomials. Verifier evaluates  $P'(r)$  and  $Q'(r)$  as the polynomials are streamed.

**Verification and output.** Looking at the final state of the EMG subroutine, Verifier constructs  $\tilde{h}(r, y)$  for all  $y \in [n^{2/3}]$  (he can treat the keys as a stream and use Fact 6). Also, from the values  $f'_j$ , he constructs  $\tilde{f}'(r, y)$  for all  $y \in [n^{2/3}]$ . The  $O(n^{1/3})$ -degree polynomial  $\tilde{g}$  is computed and stored in advance (we need to evaluate  $g$  at all points in  $[-n^{1/3}, n^{1/3}]$  and then use Lagrange interpolation to get  $\tilde{g}$ ).

Thus, Verifier can now use Equation (8) to compute  $P(r)$  and Equation (11) to compute  $Q(r)$ . He checks whether  $P(r) = P'(r)$  and  $Q(r) = Q'(r)$ . If the checks pass, he believes  $P', Q'$  are correct. He further checks whether  $\sum_{x \in [n^{1/3}]} Q'(x) = 0$ , i.e., by Equation (10), whether  $T = 0$ . If so, he believes that  $f'_j = f_j$  for all  $j \in K$ . Next, he computes  $L = \sum_{j \in K} g(f'(j))$ , and using Equation (7), he computes  $R = \sum_{x \in [n^{1/3}]} P'(x)$ . Finally,  $L + R$  gives the answer.

**Error probability.** The correctness analysis follows along standard lines of sum-check protocols. The scheme is perfectly complete since it follows from above that we always output correctly if Prover is honest. For soundness, note that the protocol fails if either  $P \neq P'$  or  $Q \neq Q'$ , but  $P(r) = P'(r)$  and  $Q(r) = Q'(r)$ . Then,  $r$  is a root of the non-zero polynomial  $P - P'$  or  $Q - Q'$ . Since degree of  $P - P'$  is  $O(n^{2/3})$  and that of  $Q - Q'$  is  $O(n^{1/3})$ , they have at most  $O(n^{2/3})$  roots in total. Since  $r$  is drawn uniformly at random from  $\mathbb{F}_q$ , where  $q > n^{p+1}$ , the probability that  $r$  is such a root is at most  $O(n^{2/3})/n^{p+1} \leq 1/\text{poly}(n)$  for sufficiently large  $n$ . Thus, the soundness error is at most  $1/\text{poly}(n)$ .

**Help and Verification costs.** The polynomials  $P$  and  $Q$  have degree  $O(n^{2/3})$  and  $O(n^{1/3})$  respectively. Thus, it requires  $O(n^{2/3} \log n)$  bits in total to express their coefficients since each coefficient comes from  $\mathbb{F}_q$  that has size  $\text{poly}(n)$ . Recall that for the setting of  $\phi = (cn^{2/3})^{-1}$ , there are  $O(\phi^{-1}) = O(n^{2/3})$  keys in the EMG algorithm. Prover sends  $f'_j$  for each  $j \in K$ , and since each frequency is at most  $m = O(n)$ , this requires  $O(n^{2/3} \log n)$  bits to communicate. Therefore, the total hcost is  $O(n^{2/3} \log n)$ .

As noted above, the invocation of EMG algorithm takes  $O(n^{2/3} \log n)$  space. Verifier maintains  $\tilde{f}(r, y)$  and stores the values  $\tilde{h}(r, y)$  and  $\tilde{f}'(r, y)$  for all  $y \in [n^{2/3}]$ . Each value is an element in  $\mathbb{F}_q$ , and hence they take up  $O(n^{2/3} \log n)$  space in total. The  $O(n^{1/3})$ -degree polynomial  $\tilde{g}$  takes  $O(n^{1/3} \log n)$  space to store. Hence, the total vcost is  $O(n^{2/3} \log n)$ .

Thus, we have proved the following theorem.

► **Theorem 1.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream of length  $m = O(n)$ . The scheme is perfectly complete and has soundness error at most  $1/\text{poly}(n)$ .*

### 3.2.2 Handling longer streams at the cost of imperfect completeness

The scheme in Section 3.2.1 requires stream length  $m = O(n)$ . Note that a turnstile stream with massive cancellations can have length  $m \gg n$ , but  $\|\mathbf{f}\|_1$  can still be  $O(n)$ . Chakrabarti et al. [7] presented their scheme under the assumption of  $m = O(n)$ , but their scheme can be made to work for longer streams as long as  $\|\mathbf{f}\|_1 = O(n)$ . We can modify our scheme to handle such streams as well without increasing the costs, but we no longer have perfect completeness. We give a sketch of this scheme below highlighting the modifications.

We cannot use the EMG algorithm anymore because it doesn't give a strong guarantee with respect to  $\|\mathbf{f}\|_1$  for turnstile streams. We use the Count-Median Sketch instead which gives the following guarantee.

► **Fact 10** (Count-Median Sketch [13]). *For all  $\phi, \varepsilon > 0$ , there exists an algorithm that, given a turnstile stream of elements in  $[n]$  with  $\|\mathbf{f}\|_1 = O(n)$ , uses  $O(\phi^{-1} \log(\varepsilon^{-1}) \log n)$  space and returns frequency estimates  $\langle \hat{f}_j : j \in [n] \rangle$  such that, with probability at least  $1 - \varepsilon$ , for all tokens  $j \in [n]$ , we have  $f_j - \phi \|\mathbf{f}\|_1 \leq \hat{f}_j \leq f_j + \phi \|\mathbf{f}\|_1$ .*

If  $\|\mathbf{f}\|_1 \leq cn$  for some constant  $c$ , then setting  $\phi = (4cn^{2/3})^{-1}$  and  $\varepsilon = 1/4$ , we get that there is an  $O(n^{2/3} \log n)$  space algorithm that, with probability at least  $3/4$ , gives

$$\forall j \in [n] : f_j - n^{1/3}/4 \leq \hat{f}_j \leq f_j + n^{1/3}/4 \quad (12)$$

For this protocol, redefine the set  $K$  as  $K := \{j : |f_j| \geq n^{1/3}/2\}$ . Prover sends a set  $K'$  that she claims is identical to  $K$ . Let  $M$  denote the set  $\{j : |\hat{f}_j| \geq 3n^{1/3}/4\}$ . Verifier checks whether  $M \subseteq K'$ , and if the check passes, he computes  $\sum_{j \in K'} g(f_j)$  and  $\sum_{j \notin K'} g(f_j)$  separately, similar to the earlier protocol, and adds them to obtain the answer.

**Error probability.** For completeness, note that if Prover is honest and  $K' = K$ , then with probability at least  $3/4$ , we have  $M \subseteq K'$ . To see this, observe that, by the guarantees of the Count-Median Sketch (Equation (12)), for all  $j \in [n]$  with  $|\hat{f}_j| \geq 3n^{1/3}/4$ , we have  $|f_j| \geq n^{1/3}/2$  with probability at least  $3/4$ . The rest of the completeness analysis is as before, and hence, there is no additional completeness error. Thus, the total completeness error of the scheme is at most  $1/4$ .

For soundness, suppose that  $K' \neq K$ . By the guarantees of the Count-Median Sketch, for all  $j \in [n]$  with  $|f_j| \geq n^{1/3}$ , we have  $|\hat{f}_j| \geq 3n^{1/3}/4$  with probability at least  $3/4$ . Thus,  $\{j : |f_j| \geq n^{1/3}\} \subseteq M$ . Hence, if the check  $M \subseteq K'$  passes, then with probability at least  $3/4$ , we have  $\{j : |f_j| \geq n^{1/3}\} \subseteq K'$ . Thus, if  $j \notin K'$ , we have  $|f_j| < n^{1/3}$ . Therefore, the computation of  $\sum_{j \notin K'} g(f_j)$  goes through as before. The additional soundness error is at most  $1/\text{poly}(n)$  as analyzed earlier. Thus, the total soundness error of the protocol is at most  $1/4 + 1/\text{poly}(n) < 1/3$ .

**Help and Verification costs.** Clearly, since  $\|\mathbf{f}\|_1 \leq cn$ , we have  $|K| = O(n^{2/3})$  which adds  $O(n^{2/3} \log n)$  bits to the hcost. The Count-Median Sketch takes space  $O(n^{2/3} \log n)$ , similar to the EMG algorithm. The rest of the cost analysis is as before, and hence we have an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme.

Thus, we have the following theorem.

► **Theorem 2.** *There is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing any frequency-based function in any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ . The scheme has completeness and soundness errors at most  $1/3$ .*

► **Remark 11.** We compare the schemes for Theorem 1 and Theorem 2 (call them Scheme 1 and Scheme 2 respectively). Scheme 2 works for streams of length  $m \gg n$  as long as  $\|\mathbf{f}\|_1 = O(n)$ , while Scheme 1 requires  $m = O(n)$ . On the negative side, Scheme 2 has imperfect completeness contrary to Scheme 1. Furthermore, the space dependence on the error  $\varepsilon$  for Scheme 2 is worse than Scheme 1: given any  $\varepsilon$ , Scheme 2 uses  $O(n^{2/3} \log n \log(\varepsilon^{-1}))$  space to bound the completeness and soundness errors by at most  $\varepsilon$ , while Scheme 1 takes  $O(n^{2/3}(\log n + \log(\varepsilon^{-1})))$  space to bound the soundness error by  $\varepsilon$ . This means that to bound the error by  $1/\text{poly}(n)$ , Scheme 2 takes  $O(n^{2/3} \log^2 n)$  space, making it weaker (though simpler) than the scheme of Chakrabarti et al. [7], which takes  $O(n^{2/3} \log^{4/3} n)$  space for the same and is also perfectly complete. For this, Scheme 1 takes only  $O(n^{2/3} \log n)$  space.

### 3.3 Special Instances and Applications

Here, we note important implications of Theorems 1 and 2. They can be applied to get similar results for multiple well-studied problems such as computing the number of distinct elements in the stream ( $F_0$ ), the highest frequency of an element in the stream ( $F_\infty$ ), and checking multiset inclusions. Note that for these problems, to the best of our knowledge, the best-known schemes were  $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$ -schemes obtained by direct application of the general scheme. Hence, we improve the bounds and simplify the schemes for these problems as well.

As a direct corollary of Theorems 1 and 2, we get the same bounds for  $F_0$ . It is an extensively studied problem in both basic streaming and stream verification. It is the special case of frequency-based functions where the function  $g$  is defined as  $g(x) = 0$  if  $x = 0$ , and  $g(x) = 1$  otherwise. Therefore, we obtain the following result.

► **Corollary 3.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_0$ , the number of distinct elements with non-zero frequency, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

Another well-studied problem related to frequency-based functions is computing  $F_\infty$ . Unlike  $F_0$ , it is not a direct special case, but a protocol for it follows by easily applying a scheme for frequency-based functions. Chakrabarti et al. [7] noted one way in which it can be applied to solve  $F_\infty$ . Here, we note a slightly alternate way which doesn't use a subroutine that their scheme uses and is tailored to our protocols: Prover sends the element  $j^* \in [n]$  that she claims has the highest frequency and a value  $f'_{j^*}$  that she claims to be equal to  $f_{j^*}$ . By the above protocols, Verifier can check whether  $f'_{j^*} = f_{j^*}$ . If the check passes, he computes  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  using the scheme above, where  $g$  is defined as  $g(x) = 0$  if  $x \leq f'_{j^*}$  and  $g(x) = 1$  otherwise. He accepts Prover's claim if  $G(\mathbf{f}) = 0$ . Thus, we get the following result.

► **Corollary 4.** *For any turnstile stream with  $\|\mathbf{f}\|_1 = O(n)$ , there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for computing  $F_\infty$ , the maximum frequency of an element, with completeness and soundness errors at most  $1/3$ . The scheme can be made perfectly complete with soundness error  $1/\text{poly}(n)$  if the stream has length  $m = O(n)$ .*

The problem of checking multiset inclusion has two multisets arriving in a stream arbitrarily interleaved between each other, and we need to check if one of them is contained in the other. This abstract problem is used as a subroutine in several other problems, e.g.,

some graph problems considered in the annotated settings [7, 9, 10]. Thus, an improved scheme for multiset inclusion implies improved subroutines for the corresponding problems. It can be solved by easy application of frequency-based functions. The reduction is already noted in Chakrabarti et al. [7], but we repeat it here for the sake of completeness.

► **Corollary 5.** *Let  $X, Y \subseteq [n]$  be multisets of size  $O(n)$ . Given a stream where elements of  $X$  and  $Y$  arrive in interleaved manner, there is an  $(n^{2/3} \log n, n^{2/3} \log n)$ -scheme for determining whether  $X \subseteq Y$ .*

**Proof.** Think of  $X$  and  $Y$  as  $n$ -length characteristic vector representations of the multisets (with an entry denoting the multiplicity of the corresponding element). Then,  $X \subseteq Y$  if and only if  $X_j \leq Y_j$  for each  $j \in [n]$ . As the elements arrive, we increment an entry if belongs to  $Y$  and decrement it if it belongs to  $X$ . Thus, the vector  $\mathbf{f}$  is given by  $f_j = Y_j - X_j$ . Define  $g$  as  $g(x) = 0$  if  $x \geq 0$  and  $g(x) = 1$  otherwise. Therefore, computing  $G(\mathbf{f}) := \sum_{j=1}^n g(f_j)$  and checking if it equals 0 solves the problem. The multisets having size  $O(n)$  ensures that the length of the stream is  $O(n)$ , and so we can safely apply our scheme. ◀

## 4 Conclusions and Open Problems

In this work, we designed two new schemes for the broad class of frequency-based functions. These schemes are much simpler than the previously best scheme known for the problem, and furthermore, they even improve upon the complexity bound for space usage and proof-size from  $O(n^{2/3} \log^{4/3} n)$  to  $O(n^{2/3} \log n)$ . The best known upper bound for prescient schemes, given by Chakrabarti et al. [7], is  $O(n^{2/3} \log n)$  for both of these complexity parameters. Hence, we also close this small gap between the prescient and online scheme complexities, and show that prescience is not necessary here to bring the polylogarithmic factor down to  $\log n$ . Additionally, the high-level framework used in our schemes is generic enough to be applicable for multiple other problems.

An important open problem in this area is to determine the asymptotic complexity of the general problem of computing frequency-based functions in the stream verification settings. Chakrabarti et al. [7] showed that any online or prescient  $(h, v)$ -scheme for the problem requires  $hv \geq n$ . Note that this lower bound leaves open the possibility of a  $(\sqrt{n}, \sqrt{n})$ -scheme, while the best known scheme achieves  $(\tilde{O}(n^{2/3}), \tilde{O}(n^{2/3}))$  for both online and prescient settings. Can we match the lower bound (up to polylogarithmic factors) and get an  $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$ -scheme for the problem, even if prescient? What about for even special cases like  $F_0$  or  $F_\infty$ ? Recall that there exist such online schemes for the  $k$ th frequency moment for some constant  $k \in \mathbb{Z}^+$  [7]. Also, it is possible to get such a scheme for  $F_0$  if we allow multiple rounds of interaction [17]. Any strict improvement on the lower bound would be extremely interesting and a breakthrough. Currently, we don't know of a function in the turnstile streaming model for which any online  $(h, v)$ -scheme must have total cost  $h + v \geq \omega(\sqrt{N})$  where  $N$  is the lower bound on its basic streaming complexity. This is related to the major open question of breaking the " $\sqrt{N}$  barrier" for the Merlin-Arthur (MA) communication model.

---

## References

- 1 Amirali Abdullah, Samira Daruki, Chitradeep Dutta Roy, and Suresh Venkatasubramanian. Streaming verification of graph properties. In *Proc. 27th International Symposium on Algorithms and Computation*, pages 3:1–3:14, 2016.

- 2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Preliminary version in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.
- 3 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.
- 4 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.
- 5 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 128–137, 2002.
- 6 Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–706, 2014.
- 7 Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Alg.*, 11(1):Article 7, 2014.
- 8 Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and Arthur-Merlin communication. In *Proc. 30th Annual IEEE Conference on Computational Complexity*, pages 217–243, 2015.
- 9 Amit Chakrabarti and Prantar Ghosh. Streaming verification of graph computations via graph structure. In *Proc. 33rd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 70:1–70:20, 2019.
- 10 Amit Chakrabarti, Prantar Ghosh, and Justin Thaler. Streaming verification for graph problems: Optimal tradeoffs and nonlinear sketches. *To appear in RANDOM*, 2020. [arXiv:2007.03039](https://arxiv.org/abs/2007.03039).
- 11 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. In *Proc. 18th Annual European Symposium on Algorithms*, pages 231–242, 2010.
- 12 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.
- 13 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Alg.*, 55(1):58–75, 2005. Preliminary version in *Proc. 6th Latin American Theoretical Informatics Symposium*, pages 29–38, 2004.
- 14 Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endowment*, 5(1):25–36, 2011.
- 15 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
- 16 Sumit Ganguly and Graham Cormode. On estimating frequency moments of data streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM 2007*, volume 4627 of *Lecture Notes in Computer Science*, pages 479–493, 2007.
- 17 Tom Gur and Ran Raz. Arthur-Merlin streaming complexity. In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 528–539, 2013.
- 18 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proc. 37th Annual ACM Symposium on the Theory of Computing*, pages 202–208, 2005.
- 19 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1161–1178, 2010.

- 20 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. 29th ACM Symposium on Principles of Database Systems*, pages 41–52, 2010.
- 21 Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In *Proc. 4th Conference on Innovations in Theoretical Computer Science*, pages 305–320, 2013.
- 22 Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP)*, volume 8572 of *LNCS*, pages 919–930, 2014.
- 23 Feifei Li, Ke Yi, Marios Hadjieleftheriou, and George Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *Proc. 33rd International Conference on Very Large Data Bases*, pages 147–158, 2007.
- 24 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.
- 25 Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.
- 26 Jelani Nelson, Huy L. Nguyễn, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012*, volume 7408 of *Lecture Notes in Computer Science*, pages 627–638, 2012.
- 27 Stavros Papadopoulos, Yin Yang, and Dimitris Papadias. Cads: Continuous authentication on data streams. In *Proc. 33rd International Conference on Very Large Data Bases*, pages 135–146, 2007.
- 28 Adi Shamir.  $IP = PSPACE$ . *J. ACM*, 39(4):869–877, 1992.
- 29 Justin Thaler. Data stream verification. In *Encyclopedia of Algorithms*, pages 494–499. Springer Berlin Heidelberg, 2016.
- 30 Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *Proc. 43rd International Colloquium on Automata, Languages and Programming*, pages 59:1–59:14, 2016.
- 31 Peter A. Tucker, David Maier, Lois M. L. Delcambre, Tim Sheard, Jennifer Widom, and Mark P. Jones. Punctuated data streams, 2005.
- 32 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 167–175, 2004.
- 33 Ke Yi, Feifei Li, Marios Hadjieleftheriou, George Kollios, and Divesh Srivastava. Randomized synopses for query assurance on data streams. In *Proc. 24th International Conference on Data Engineering*, pages 416–425, 2008.



# Online Carpooling Using Expander Decompositions

**Anupam Gupta**

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA  
anupamg@cs.cmu.edu

**Ravishankar Krishnaswamy**

Microsoft Research, Bengaluru, India  
rakri@microsoft.com

**Amit Kumar**

Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, India  
amitk@cse.iitd.ernet.in

**Sahil Singla**

Department of Computer Science, Princeton University, NJ, USA  
singla@cs.princeton.edu

---

## Abstract

We consider the online carpooling problem: given  $n$  vertices, a sequence of edges arrive over time. When an edge  $e_t = (u_t, v_t)$  arrives at time step  $t$ , the algorithm must orient the edge either as  $v_t \rightarrow u_t$  or  $u_t \rightarrow v_t$ , with the objective of minimizing the maximum discrepancy of any vertex, i.e., the absolute difference between its in-degree and out-degree. Edges correspond to pairs of persons wanting to ride together, and orienting denotes designating the driver. The discrepancy objective then corresponds to every person driving close to their fair share of rides they participate in.

In this paper, we design efficient algorithms which can maintain  $\text{polylog}(n, T)$  maximum discrepancy (w.h.p) over any sequence of  $T$  arrivals, when the arriving edges are sampled independently and uniformly from any given graph  $G$ . This provides the first polylogarithmic bounds for the online (stochastic) carpooling problem. Prior to this work, the best known bounds were  $O(\sqrt{n \log n})$ -discrepancy for any adversarial sequence of arrivals, or  $O(\log \log n)$ -discrepancy bounds for the stochastic arrivals when  $G$  is the complete graph.

The technical crux of our paper is in showing that the simple greedy algorithm, which has provably good discrepancy bounds when the arriving edges are drawn uniformly at random from the complete graph, also has polylog discrepancy when  $G$  is an expander graph. We then combine this with known expander-decomposition results to design our overall algorithm.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Online algorithms

**Keywords and phrases** Online Algorithms, Discrepancy Minimization, Carpooling

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.23

**Related Version** <https://arxiv.org/abs/2007.10545>.

**Funding** This research was done under the auspices of the Indo-US Virtual Networked Joint Center IUSSTF/JC-017/2017.

*Anupam Gupta*: Supported in part by NSF awards CCF-1907820, CCF1955785, and CCF-2006953.

*Sahil Singla*: Supported in part by the Schmidt Foundation.

**Acknowledgements** We thank Thatchaphol Saranurak for explaining and pointing us to [7, Theorem 5.6]. The last author would like to thank Navin Goyal for introducing him to [1].



© Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Sahil Singla;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 23; pp. 23:1–23:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Consider the following *edge orientation* problem: we are given a set  $V$  of  $n$  nodes, and undirected edges arrive online one-by-one. Upon arrival of an edge  $\{u, v\}$ , it has to be oriented as either  $u \rightarrow v$  or  $v \rightarrow u$ , immediately and irrevocably. The goal is to minimize the *discrepancy* of this orientation at any time  $t \in [T]$  during the arrival process, i.e., the maximum imbalance between the in-degree and out-degree of any node. Formally, if we let  $\chi^t$  to denote the orientation at time  $t$  and  $\delta_t^-(v)$  (resp.  $\delta_t^+(v)$ ) to denote the number of in-edges (resp. out-edges) incident to  $v$  in  $\chi^t$ , then we want to minimize

$$\max_t \text{disc}(\chi^t) := \max_t \max_v |\delta_t^-(v) - \delta_t^+(v)|.$$

If the entire sequence of edges is known up-front, one can use a simple cycle-and-path-peeling argument to show that any set of edges admit a discrepancy of at most 1. The main focus of this work is in understanding how much loss is caused by the presence of uncertainty, since we don't have knowledge of future arrivals when we irrevocably orient an edge.

This problem was proposed by Ajtai et al. [1] as a special case of the *carpooling problem* where hyperedges arrive online, each representing a carpool where one person must be designated as a driver. The “fair share” of driving for person  $i$  can be defined as  $\sum_{e:i \in e} 1/|e|$ , and we would like each person to drive approximately this many times. In the case of graphs where each carpool is of size  $|e| = 2$ , this carpooling problem is easily transformed into the edge-orientation problem.

Ajtai et al. showed that while deterministic algorithms cannot have an  $o(n)$  discrepancy, they gave a randomized “local greedy” which has an expected discrepancy (for any  $T \geq 1$ ) of  $O(\sqrt{n \log n})$  for any online input sequence of  $T$  arrivals. Indeed, note that the discrepancy bound is independent of the length of the sequence  $T$ , and depends only on the number of nodes, thus giving a non-trivial improvement over the naive random assignment, which will incur a discrepancy of  $O(\sqrt{T \log n})$ . Intriguingly, the lower bound they show for online algorithms is only  $\Omega((\log n)^{1/3})$  – leaving a large gap between the upper and lower bounds.

Given its apparent difficulty in the adversarial online model, Ajtai et al. proposed a stochastic model, where each edge is an independent draw from some underlying probability distribution over pairs of vertices. They considered the the uniform distribution, which is the same as presenting a uniformly random edge of the complete graph at each time. In this special case, they showed that the greedy algorithm (which orients each edge towards the endpoint with lower in-degree minus out-degree) has expected discrepancy  $\Theta(\log \log n)$ . Their analysis crucially relies on the structure and symmetry of the complete graph.

In this paper, we consider this stochastic version of the problem for general graphs: i.e., given an arbitrary simple graph  $G$ , the online input is a sequence of edges chosen independently and uniformly at random (with replacement) from the edges of this graph  $G^1$ . Our main result is the following:

► **Theorem 1 (Main Theorem).** *There is an efficient algorithm for the edge-orientation problem that maintains, w.h.p, a maximum discrepancy of  $O(\text{poly} \log(nT))$  on input sequences formed by i.i.d. draws from the edges of a given graph  $G$ .*

<sup>1</sup> It is possible to extend our results, by losing a  $\log T$  factor, to edge-weighted distributions where an edge is drawn i.i.d. with probability proportional to its weight. Since this extension uses standard ideas like bucketing edges with similar weights, we restrict our attention to arrivals from a graph  $G$  for simplicity.

## 1.1 Our Techniques

Let us fix some notation. Given a (multi)graph  $G = (V, E)$  with  $|V| = n$ , the algorithm is presented with a vector  $v^t$  at each time as follows. A uniformly random edge  $(u, v) \in G$  is sampled, and the associated characteristic vector  $v^t = \mathbf{e}_u - \mathbf{e}_v$  is presented to the algorithm, where  $\mathbf{e}_u \in \mathbb{R}^n$  has all zeros except index  $u$  being 1. The algorithm must immediately sign  $v^t$  with  $\chi^t \in \{-1, 1\}$ , to keep the discrepancy bounded at all times  $t$ . Here the discrepancy of node  $u$  at time  $t$  is the  $u^{\text{th}}$  entry of the vector  $\sum_{s \leq t} \chi^s v^s$  (which could be negative), and the discrepancy of the algorithm is the maximum absolute discrepancy over all vertices, i.e.,  $\left\| \sum_{s \leq t} \chi^s v^s \right\|_{\infty}$ .

A natural algorithm is to pick a uniformly random orientation for each arriving edge. This maintains zero expected discrepancy at each node. However, the large variance may cause the maximum discrepancy over nodes to be as large as  $\Omega(\sqrt{T})$ , where  $T$  the total number of edges (which is the same as the number of time-steps). For example, this happens even on  $T$  parallel edges between two nodes. In this case, however, the *greedy algorithm* which orients the edge from the vertex of larger discrepancy to that of smaller discrepancy works well. Indeed it is not known to be bad for stochastic instances. (Since it is a deterministic algorithm, it can perform poorly on adversarial inputs due to known  $o(n)$  lower bounds [1].)

Building on the work of Ajtai et al. who consider stochastic arrivals on complete graphs, the first step towards our overall algorithm is to consider the problem on *expander graphs*. At a high level, one hurdle to achieving low discrepancy in the stochastic case is that we reach states where both endpoints of a randomly-chosen edge already have equally high discrepancy. Then, no matter how we orient the edge, we increase the maximum discrepancy. But this should not happen in expander graphs: if  $S$  is the set of “high” discrepancy vertices, then the expansion of the graph implies that  $|\partial S|$  must be a large fraction of the total number of edges incident to  $S$ . Therefore, intuitively, we have a good chance of reducing the discrepancy if we get edges that go from  $S$  to low-degree nodes. To make this idea formal, we relate the greedy process on expander graphs  $G$  to the so-called  $(1 + \beta)$ -process over an *easier arrival* sequence where the end-points of a new edge are chosen from a *product distribution*, where the probability of choosing a vertex is proportional to its degree in  $G$ . However, in the  $(1 + \beta)$ -process<sup>2</sup>, the algorithm orients a new edge greedily with only probability  $\beta$  for some small value of  $\beta$ , and does a random orientation with the remaining probability  $(1 - \beta)$ .

Indeed, we compare these two processes by showing that (a) the expected increase of a natural potential  $\Phi := \sum_v \cosh(\lambda \text{discrepancy}(v))$  – which can be thought of as a soft-max function – is lower for the greedy algorithm on expanders when compared to the  $(1 + \beta)$ -process on the product distribution, and (b) the same potential increases very slowly (if at all) on the product distribution. A similar idea was used by Peres et al. [13] for a related stochastic load balancing problem; however, many of the technical details are different.

The second component of the algorithm is to decompose a general graph into expanders. This uses the (by-now commonly used) idea of expander decompositions. Loosely speaking, this says that the edges of any graph can be decomposed into some number of smaller graphs (each being defined on some subset of vertices), such that (a) each of these graphs is an expander, and (b) each vertex appears in only a poly-logarithmic number of these expanders. Our arguments for expanders require certain weak-regularity properties – namely the degrees of vertices should not be too small compared to the average degree – and hence some care is required in obtaining decompositions into such expanders. These details appear in the full version.

<sup>2</sup> The name  $(1 + \beta)$ -process stems from the notion for an analogous load-balancing (or) balls-and-bins setting [13], this process would be like the  $(1 + \beta)$ -fractional version of the power-of-two choices process.

Our overall algorithm can then be summarized in Algorithm 1.

■ **Algorithm 1 DivideAndGreedy** (graph  $G = (V, E)$ ).

- 
- 1: run the expander-decomposition algorithm in Theorem 19 (in Section 2.5) on  $G$  to obtain a collection  $\mathcal{P} = \{G_1, G_2, \dots, G_k\}$  of edge-disjoint expander graphs.
  - 2: initialize  $\mathcal{H} = \{H_1, H_2, \dots, H_k\}$  to be a collection of empty graphs, where  $H_i$  is the directed multi-graph consisting of all edges which have arrived corresponding to base graph  $G_i$ , along with their orientations assigned by the algorithm upon arrival.
  - 3: **for** each new edge  $e \equiv \{u, v\}$  that arrives at time-step  $t$  **do**
  - 4:   let  $i$  denote the index such that  $e \in G_i$  according to our decomposition.
  - 5:   add  $e$  to  $H_i$ , and orient  $e$  in a greedy manner w.r.t  $H_i$ , i.e., from  $u$  to  $v$  if  $\text{disc}_{H_i}(u) \geq \text{disc}_{H_i}(v)$ , where  $\text{disc}_H(w) = \delta_{H_i}^{\text{in}}(w) - \delta_{H_i}^{\text{out}}(w)$  is the in-degree minus out-degree of any vertex  $w$  in the current sub-graph  $H_i$  maintained by the algorithm.
  - 6: **end for**
- 

## 1.2 Related Work

The study of discrepancy problems has a long history; see the books [12, 8] for details on the classical work. The problem of online discrepancy minimization was studied by Spencer [14], who showed an  $\Omega(\sqrt{T})$  lower bound for for adaptive adversarial arrivals. More refined lower bounds were given by Bárány [6]; see [4] for many other references. Much more recently, Bansal and Spencer [5] and Bansal et al. [4] consider a more general vector-balancing problem, where each request is a vector  $v^t \in \mathbb{R}^n$  with  $\|v^t\|_\infty \leq 1$ , and the goal is to assign a sign  $\chi^t \in \{-1, 1\}$  to each vector to minimize  $\|\sum_t \chi^t v^t\|_\infty$ , i.e., the largest coordinate of the signed sum. Imagining each edge  $e_t = \{u, v\}$  to be the vector  $\frac{1}{\sqrt{2}}(\mathbf{e}_u - \mathbf{e}_v)$  (where this initial sign is chosen arbitrarily) captures the edge-orientation problem up to constant factors. Bansal et al. gave an  $O(n^2 \log(nT))$ -discrepancy algorithm for the natural stochastic version of the problem under general distributions. For some special geometric problems, they gave an algorithm that maintains  $\text{poly}(s, \log T, \log n)$  discrepancy for sparse vectors that have only  $s$  non-zero coordinates. These improve on the work of Jiang et al. [11], who give a sub-polynomial discrepancy coloring for online arrivals of points on a line. A related variant of these geometric problems was also studied in Dwivedi et al. [9].

Very recently, an independent and exciting work of Alweiss, Liu, and Sawhney [2] gave a randomized algorithm that maintains a discrepancy of  $O(\log(nT)/\delta)$  for any input sequence chosen by an oblivious adversary with probability  $1 - \delta$ , even for the more general vector-balancing problem for vectors of unit Euclidean norm (the so-called Kómlós setting). Instead of a potential based analysis like ours, they directly argue why a carefully chosen randomized greedy algorithm ensures w.h.p. that the discrepancy vector is always sub-Gaussian. A concurrent work of Bansal et al. [3] also obtains similar results for i.i.d. arrivals, but they use a very different potential than our expander-decomposition approach. It is an interesting open question to extend our approach to hypergraphs and re-derive their results.

## 1.3 Notation

We now define some graph-theoretic terms that are useful for the remainder of the paper.

► **Definition 2** (Volume and  $\alpha$ -expansion). Given any graph  $G = (V, E)$ , and set  $S \subseteq V$  its *volume* is defined to be  $\text{vol}(S) := \sum_{v \in S} \text{degree}(v)$ . We say  $G$  is an  $\alpha$ -*expander* if

$$\min_{S \subseteq V} \frac{|E(S, V \setminus S)|}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}} \geq \alpha.$$

We will also need the following definition of “weakly-regular” graphs, which are graphs where every vertex has degree *at least* a constant factor of the average degree. Note that the *maximum* degree can be arbitrarily larger than the average degree.

► **Definition 3** ( $\gamma$ -weakly-regular). For  $\gamma \in [0, 1]$ , a graph  $G = (V, E)$  is called  $\gamma$ -*weakly-regular* if every vertex  $v \in V$  has degree at least  $\gamma \cdot \sum_{u \in V} \text{degree}(u) / |V|$ .

► **Definition 4** (Discrepancy Vector). Given any directed graph  $H = (V, A)$  (representing all the oriented edges until any particular time-step), let  $\mathbf{d} \in \mathbb{Z}^{|V|}$  represent the discrepancy vector of the current graph, i.e. the  $v^{\text{th}}$  entry of  $\mathbf{d}$ , denoted by  $d_v$  is the difference between the number of in-edges incident at  $v$  and the number of out-edges incident at  $v$  in  $H$ .

## 2 The Greedy Algorithm on Expander Graphs

In this section, we consider the special case when the graph  $G$  is an expander. More formally, we show that the greedy algorithm is actually good for such graphs.

► **Definition 5** (Expander Greedy Process). The greedy algorithm maintains a current discrepancy  $d_v^t$  for each vertex  $v$ , which is the in-degree minus out-degree of every vertex among the previously arrived edges. Initially,  $d_v^1 = 0$  for every vertex  $v$  at the beginning of time-step 1. At each time  $t \geq 1$ , a uniformly random edge  $e \in G$  with end-points  $\{u, v\}$  is presented to the algorithm, and suppose w.l.o.g.  $d_u^t \geq d_v^t$ , i.e.,  $u$  has larger discrepancy (ties broken arbitrarily). Then, the algorithm orients the edge from  $u$  to  $v$ . The discrepancies of  $u$  and  $v$  become  $d_u^{t+1} = d_u^t - 1$  and  $d_v^{t+1} = d_v^t + 1$ , and other vertices’ discrepancies are unchanged.

► **Theorem 6.** Consider any  $\gamma$ -weakly-regular  $\alpha$ -expander  $G$ , and suppose edges are arriving as independent samples from  $G$  over a horizon of  $T$  time-steps. Then, the greedy algorithm maintains a discrepancy  $d_v^t$  of  $O(\log^5 nT)$  for every time  $t$  in  $[0 \dots T]$  and every vertex  $v$ , as long as  $\alpha \geq 6\lambda$ ,  $\gamma \geq \lambda^{1/4}$ , where  $\lambda = O(\log^{-4} nT)$ .

For the sake of concreteness, it might be instructive to assume  $\alpha \approx \gamma \approx O(\frac{1}{\log n})$ , which is roughly what we will obtain from our expander-decomposition process.

### 2.1 Setting Up The Proof

Our main idea is to introduce *another random process* called the  $(1+\beta)$ -process, and show that the  $(1+\beta)$ -process stochastically dominates the expander-greedy process in a certain manner, and separately bound the behaviour of the  $(1+\beta)$ -process subsequently. By combining these two, we get our overall analysis of the expander-greedy process.

To this end, we first define a random arrival sequence where the end-points of each new edge are actually sampled independently from a *product distribution*.

► **Definition 7** (Product Distribution). Given a set  $V$  of vertices with associated weights  $\{w_v \geq 0 \mid v \in V\}$ , at each time  $t$ , we select two vertices  $u, v$  as *two independent samples* from  $V$ , according to the distribution where any vertex  $v \in V$  is chosen with probability  $\frac{w_v}{\sum_{v' \in V} w_{v'}}$ , and the vector  $v^t := \chi_u - \chi_v$  is presented to the algorithm.

We next define the  $(1 + \beta)$ -process, which will be crucial for the analysis.

► **Definition 8** ( $(1 + \beta)$ -process on product distributions). Consider a product distribution over a set of vertices  $V$ . When presented with a vector  $v^t := \chi_u - \chi_v$  from this product distribution at time  $t$ , the  $(1 + \beta)$ -process assigns a sign to the vector  $v^t$  as follows: with probability  $(1 - \beta)$ , it assigns it uniformly  $\pm 1$ , and only with the remaining probability  $\beta$  it uses the greedy algorithm to sign this vector.

Note that setting  $\beta = 1$  gives us back the greedy algorithm, and  $\beta = 0$  gives an algorithm that assigns a random sign to each vector.

► **Remark 9.** The original  $(1 + \beta)$ -process was in fact introduced in [13], where Peres et al. analyzed a general load-balancing process over  $n$  bins (corresponding to vertices), and balls arrive sequentially. Upon each arrival, the algorithm gets to sample a random edge from a  $k$ -regular expander<sup>3</sup>  $G$  over the bins, and places the ball in the lighter loaded bin among the two end-points of the edge. They show that this process maintains a small maximum load, by relating it to an analogous  $(1 + \beta)$ -process, where instead of sampling an edge from  $G$ , two bins are chosen uniformly at random, and the algorithm places the ball into a random bin with probability  $1 - \beta$ , and the lesser loaded bin with probability  $\beta$ . Note that their analysis inherently assumed that the two vertices are sampled from the uniform distribution where all weights  $w_u$  are equal. By considering arbitrary product distributions, we are able to handle arbitrary graphs with a non-trivial conductance, i.e., even those that *do not* satisfy the  $k$ -regularity property. This is crucial for us because the expander decomposition algorithms, which reduce general graphs to a collection of expanders, do not output regular expanders.

Our analysis will also involve a potential function (intuitively the soft-max of the vertex discrepancies) for both the expander-greedy process as well as the  $(1 + \beta)$ -process.

► **Definition 10** (Potential Function). Given vertex discrepancies  $\mathbf{d} \in \mathbb{Z}^{|V|}$ , define

$$\Phi(\mathbf{d}) := \sum_v \cosh(\lambda d_v), \tag{1}$$

where  $\lambda < 1$  is a suitable parameter to be optimized.

Following many prior works, we use the hyperbolic cosine function to symmetrize for positive and negative discrepancy values. When  $\mathbf{d}$  is clear from the context, we will write  $\Phi(\mathbf{d})$  as  $\Phi$ . We will also use  $\mathbf{d}^t$  to refer to the discrepancy vector at time  $t$ , and  $d_u^t$  to the discrepancy of  $u$  at time  $t$ . We will often ignore the superscript  $t$  if it is clear from the context.

We are now ready to define the appropriate parameters of the  $(1 + \beta)$ -process. Indeed, given the expander-greedy process defined on graph  $G$ , we construct an associated  $(1 + \beta)$ -process where for each vertex  $v$ , the probability of sampling any vertex in the product distribution is proportional to its degree in  $G$ , i.e.,  $w_v = \text{degree}_G(v)$  for all  $v \in V$ . We also set the  $\beta$  parameter equal to  $\alpha$ , the conductance of the graph  $G$ .

## 2.2 One-Step Change in Potential

The main idea of the proof is to use a *majorization argument* to argue that *the expected one-step change* in potential of the expander process can be upper bounded by that of the  $(1 + \beta)$ -process, if the two processes start at the same discrepancy configuration  $\mathbf{d}^t$ . Subsequently, we bound the one-step change for the  $(1 + \beta)$ -process in Section 2.4.

<sup>3</sup> Actually their proof works for a slightly more general notion of expanders, but which is still insufficient for our purpose.

To this end, consider a time-step  $t$ , where the current discrepancy vector of the expander process is  $\mathbf{d}^t$ . Suppose the next edge in the expander process is  $(i, j)$ , where  $d_i^t > d_j^t$ . Then the greedy algorithm will always choose a sign such that  $d_i$  decreases by 1, and  $d_j$  increases by 1. Indeed, this ensures the overall potential is non-increasing unless  $d_i = d_j$ . More importantly, the potential term for other vertices remains unchanged, and so we can express the expected change in potential as having contributions from precisely two terms, one due to  $d_i \rightarrow d_i - 1$  (called the *decrease term*), and denoted as  $\Delta_{-1}(t)$ , and one due to  $d_j \rightarrow d_j + 1$  (the *increase term*), denoted as  $\Delta_{+1}(t)$ :

$$\begin{aligned} \mathbb{E}_{(i,j) \sim G}[\Delta\Phi] &= \mathbb{E}_{(i,j) \sim G}[\Phi(\mathbf{d}^{t+1}) - \Phi(\mathbf{d}^t)] \\ &= \underbrace{\mathbb{E}_{(i,j)}[\cosh(\lambda(d_i - 1)) - \cosh(\lambda(d_i))]}_{=:\Delta_{-1}(\mathbf{d}^t)} + \underbrace{\mathbb{E}_{(i,j)}[\cosh(\lambda(d_j + 1)) - \cosh(\lambda(d_j))]}_{=:\Delta_{+1}(\mathbf{d}^t)}. \end{aligned}$$

Now, consider the  $(1 + \beta)$ -process on the vertex set  $V$ , where the product distribution is given by weights  $w_u = \deg(u)$  for each  $u \in V$ , starting with the same discrepancy vector  $\mathbf{d}^t$  as the expander process at time  $t$ . Then, if  $u$  and  $v$  are the two vertices sampled independently according to the product distribution, then by its definition, the  $(1 + \beta)$ -process signs this pair randomly with probability  $(1 - \beta)$ , and greedily with probability  $\beta$ . For the sake of analysis, we define two terms analogous to  $\Delta_{-1}(\mathbf{d}^t)$  and  $\Delta_{+1}(\mathbf{d}^t)$  for the  $(1 + \beta)$ -process. To this end, let  $i \in \{u, v\}$  denote the identity of the random vertex to which the  $(1 + \beta)$ -process assigns  $+1$ . Define

$$\tilde{\Delta}_{+1}(\mathbf{d}^t) := \mathbb{E}_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[\cosh(\lambda(d_i + 1)) - \cosh(\lambda(d_i))], \quad (2)$$

where  $\mathbf{w} \times \mathbf{w}$  refers to two independent choices from the product distribution corresponding to  $w$ . Similarly let  $j \in \{u, v\}$  denote the identity of the random vertex to which the  $(1 + \beta)$ -process assigns  $-1$ , and define

$$\tilde{\Delta}_{-1}(\mathbf{d}^t) := \mathbb{E}_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[\cosh(\lambda(d_j - 1)) - \cosh(\lambda(d_j))]. \quad (3)$$

In what follows, we bound  $\Delta_{-1}(\mathbf{d}^t) \leq \tilde{\Delta}_{-1}(\mathbf{d}^t)$  through a coupling argument, and similarly bound  $\Delta_{+1}(\mathbf{d}^t) \leq \tilde{\Delta}_{+1}(\mathbf{d}^t)$  using a separate coupling.

A subtlety: the expected one-step change in  $\Phi$  in the expander process precisely equals  $\Delta_{-1}(\mathbf{d}^t) + \Delta_{+1}(\mathbf{d}^t)$ . However, if we define an analogous potential for the  $(1 + \beta)$ -process, then the one-step change in potential there *does not* equal the sum  $\tilde{\Delta}_{-1}(\mathbf{d}^t) + \tilde{\Delta}_{+1}(\mathbf{d}^t)$ . Indeed, we sample  $u$  and  $v$  i.i.d. in the  $(1 + \beta)$ -process, it is possible that  $u = v$  and therefore the one-step change in potential is 0, while the sum  $\tilde{\Delta}_{-1}(\mathbf{d}^t) + \tilde{\Delta}_{+1}(\mathbf{d}^t)$  will be non-zero. Hence the following lemma does not bound the expected potential change for the expander process by that for the  $(1 + \beta)$ -process (both starting from the same state), but by this surrogate  $\tilde{\Delta}_{-1}(\mathbf{d}^t) + \tilde{\Delta}_{+1}(\mathbf{d}^t)$ , and it is this surrogate sum that we bound in Section 2.4.

### 2.3 The Coupling Argument

We now show a coupling between the expander-greedy process and the  $(1 + \beta)$ -process defined in Section 2.1, to bound the expected one-step change in potential for the expander process.

► **Lemma 11.** *Given an  $\alpha$ -expander  $G = (V, E)$ , let  $\mathbf{d}^t \equiv (d_v : v \in V)$  denote the current discrepancies of the vertices at any time step  $t$  for the expander-greedy process. Consider a hypothetical  $(1 + \beta)$ -process on vertex set  $V$  with  $\beta = \alpha$ , the weight of vertex  $v \in V$  set to  $w_v = \deg(v)$ , and starting from the same discrepancy state  $\mathbf{d}^t$ . Then:*

## 23:8 Online Carpooling Using Expander Decompositions

(a)  $\Delta_{-1}(\mathbf{d}^t) \leq \tilde{\Delta}_{-1}(\mathbf{d}^t)$ , and (b)  $\Delta_{+1}(\mathbf{d}^t) \leq \tilde{\Delta}_{+1}(\mathbf{d}^t)$ .  
Hence the expected one-step change in potential  $\mathbb{E}[\Phi(\mathbf{d}^{t+1}) - \Phi(\mathbf{d}^t)] \leq \tilde{\Delta}_{-1}(\mathbf{d}^t) + \tilde{\Delta}_{+1}(\mathbf{d}^t)$ .

**Proof.** We start by renaming the vertices in  $V$  such that  $d_n \leq d_{n-1} \leq \dots \leq d_1$ . Suppose the next edge in the expander process corresponds to indices  $i, j$  where  $i < j$ . We prove the lemma statement by two separate coupling arguments, which crucially depend on the following claim. Intuitively, this claim shows that a  $-1$  is more likely to appear among the high discrepancy vertices of  $G$  in the expander process than the  $(1 + \beta)$ -process (thereby having a lower potential), and similarly a  $+1$  is more likely to appear among the low discrepancy vertices of  $G$  in the expander process than in the  $(1 + \beta)$ -process. Peres et al. [13] also prove a similar claim for stochastic load balancing, but they only consider uniform distributions.

▷ **Claim 12.** For any  $k \in [n]$ , if  $S_k$  denotes the set of vertices with indices  $k' \in [k]$  (the  $k$  highest discrepancy vertices) and  $T_k$  denotes  $V \setminus S_k$ , then

$$\Pr_{(i,j) \sim G}[-1 \in S_k] \geq \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[-1 \in S_k] \quad \text{and} \quad \Pr_{(i,j) \sim G}[+1 \in T_k] \geq \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[+1 \in T_k].$$

Above, we abuse notation and use the terminology “ $-1 \in S_k$ ” to denote that the vertex whose discrepancy decreases falls in the set  $S_k$  in the corresponding process.

**Proof.** Fix an index  $k$ , and let  $\rho := \frac{\text{vol}(S_k)}{\text{vol}(V)}$  be the relative volume of  $S_k$ , i.e., the fraction of edges of  $G$  incident to the  $k$  nodes of highest degree. First we consider the  $(1 + \beta)$ -process on  $V$ . With  $(1 - \beta)$  probability we assign a sign to the input vector uniformly at random. Therefore, conditioned on this choice, a vertex in  $S_k$  will get a  $-1$  sign with probability

$$\frac{1}{2} \cdot \Pr[u \in S_k] + \frac{1}{2} \Pr[v \in S_k] = \frac{\text{vol}(S_k)}{\text{vol}(V)} = \rho,$$

where  $u$  and  $v$  denote the two vertices chosen by the  $(1 + \beta)$ -process. With probability  $\beta$ , we will use the greedy algorithm, and so  $-1$  will appear on a vertex in  $S_k$  iff at least one of the two chosen vertices lie in  $S_k$ . Putting it together, we get

$$\begin{aligned} \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[-1 \in S_k] &= (1 - \beta) \cdot \frac{\text{vol}(S_k)}{\text{vol}(V)} + \beta \cdot \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[\{u, v\} \cap S_k \neq \emptyset] \\ &= (1 - \beta) \cdot \rho + \beta \cdot (1 - (1 - \rho)^2) = (1 + \beta - \beta \cdot \rho) \cdot \rho. \end{aligned} \quad (4)$$

Now we consider the expander process. A vertex in  $S_k$  gets  $-1$  iff the chosen edge has at least one end-point in  $S_k$ . Therefore,

$$\begin{aligned} \Pr_{(i,j) \sim G}[-1 \in S_k] &= \Pr[i \in S_k] = \frac{|E(S_k, S_k)| + |E(S_k, V \setminus S_k)|}{|E|} \\ &= \frac{(2|E(S_k, S_k)| + |E(S_k, V \setminus S_k)|) + |E(S_k, V \setminus S_k)|}{2|E|} = \frac{\text{vol}(S_k) + |E(S_k, V \setminus S_k)|}{\text{vol}(V)}. \end{aligned}$$

Recalling that  $\beta = \alpha$ , and that  $G$  is an  $\alpha$ -expander, we consider two cases:

**Case 1:** If  $\text{vol}(S_k) \leq \text{vol}(V \setminus S_k)$ , we use

$$\begin{aligned} \Pr_{(i,j) \sim G}[-1 \in S_k] &= \frac{\text{vol}(S_k) + |E(S_k, V \setminus S_k)|}{\text{vol}(V)} \\ &\geq (1 + \alpha) \frac{\text{vol}(S_k)}{\text{vol}(V)} = (1 + \beta) \rho \geq \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[-1 \in S_k]. \end{aligned}$$

**Case 2:** If  $\text{vol}(S_k) > \text{vol}(V \setminus S_k)$ , we use

$$\begin{aligned} \Pr_{(i,j) \sim G}[-1 \in S_k] &= \frac{\text{vol}(S_k) + |E(S_k, V \setminus S_k)|}{\text{vol}(V)} \geq \frac{\text{vol}(S_k) + \alpha \cdot \text{vol}(V \setminus S_k)}{\text{vol}(V)} \\ &\geq \left(1 + \beta \cdot \frac{\text{vol}(V \setminus S_k)}{\text{vol}(V)}\right) \cdot \rho = \Pr_{(i,j) \sim \mathbf{w} \times \mathbf{w}}[-1 \in S_k], \end{aligned}$$

where the last equality uses (4).

This completes the proof of  $\Pr_{(i,j) \sim G}[-1 \in S_k] \geq \Pr_{(i,j) \sim \mathbf{w}}[-1 \in S_k]$ . One can similarly show  $\Pr_{(i,j) \sim G}[+1 \in T_k] \geq \Pr_{(u,v) \sim \mathbf{w} \times \mathbf{w}}[+1 \in T_k]$ , which completes the proof of the claim.  $\triangleleft$

Claim 12 shows that we can establish a coupling between the two processes such that if  $-1$  belongs to  $S_k$  in  $(1 + \beta)$ -process, then the same happens in the expander process. In other words, there is a joint sample space  $\Omega$  such that for any outcome  $\omega \in \Omega$ , if vertices  $v_a$  and  $v_b$  get sign  $-1$  in the expander process and the  $(1 + \beta)$ -process respectively, then  $a \leq b$ .

Let  $\mathbf{d}$  and  $\tilde{\mathbf{d}}$  denote the discrepancy vectors in the expander process and the  $(1 + \beta)$ -process after the  $-1$  sign has been assigned, respectively. Now, since both the processes start with the same discrepancy vector  $\mathbf{d}^t$ , we see that for any fixed outcome  $\omega \in \Omega$ , the vector  $\tilde{\mathbf{d}}$  majorizes  $\mathbf{d}$  in the following sense.

**► Definition 13 (Majorization).** Let  $\mathbf{a}$  and  $\mathbf{b}$  be two real vectors of the same length  $n$ . Let  $\vec{\mathbf{a}}$  and  $\vec{\mathbf{b}}$  denote the vectors  $\mathbf{a}$  and  $\mathbf{b}$  with coordinates rearranged in descending order respectively. We say that  $\mathbf{a}$  majorizes  $\mathbf{b}$ , written  $\mathbf{a} \succeq \mathbf{b}$ , if for all  $i$ ,  $1 \leq i \leq n$ , we have  $\sum_{j=1}^i \vec{\mathbf{a}}_j \geq \sum_{j=1}^i \vec{\mathbf{b}}_j$ .

One of the properties of majorization [10] is that any convex and symmetric function of the discrepancy vector (which  $\Phi$  is) satisfies that  $\Phi(\mathbf{d}) \leq \Phi(\tilde{\mathbf{d}})$ . Thus, for any fixed outcome  $\omega$ , the change in potential in the expander process is at most that of the surrogate potential in the  $(1 + \beta)$ -process. Since  $\Delta_{-1}(\mathbf{d}^t)$  and  $\tilde{\Delta}_{-1}(\mathbf{d}^t)$  are just the expected change of these quantities in the two processes (due to assignment of  $-1$  sign), the first statement of the lemma follows. Using an almost identical proof, we can also show the second statement. (Note that we may need to redefine the coupling between the two processes to ensure that if vertices  $v_a, v_b$  get sign  $+1$  as above, then  $b \leq a$ .)  $\blacktriangleleft$

## 2.4 Analyzing One-Step $\Delta\Phi$ of the $(1 + \beta)$ -process

Finally we bound the one-step change in (surrogate) potential of the  $(1 + \beta)$ -process starting at discrepancy vector  $\mathbf{d}^t$ ; recall the definitions of  $\tilde{\Delta}_{-1}(\mathbf{d}^t)$  and  $\tilde{\Delta}_{+1}(\mathbf{d}^t)$  from Section 2.2.

**► Lemma 14.** *If  $\Phi(\mathbf{d}^t) \leq (nT)^{10}$ , and if the weights  $w_v$  are such that for all  $v$ ,  $\frac{w_v}{\sum_{v'} w_{v'}} \geq \frac{\gamma}{n}$  (i.e., the minimum weight is at least a  $\gamma$  fraction of the average weight), then we have that*

$$\tilde{\Delta}_{-1}(\mathbf{d}^t) + \tilde{\Delta}_{+1}(\mathbf{d}^t) \leq O(1),$$

as long as  $\beta \geq 6\lambda$ ,  $\gamma \geq 16\lambda^{1/4}$ , and  $\lambda = O(\log^{-4} nT)$ .

**Proof.** Let  $u$  be an arbitrary vertex in  $V$ , and we condition on the fact that the first vertex chosen by the  $(1 + \beta)$ -process is  $u$ . Then, we show that

$$\mathbb{E}_{v \sim \mathbf{w}} \left[ \cosh(\lambda(d_i - 1)) - \cosh(\lambda(d_i)) + \cosh(\lambda(d_j + 1)) - \cosh(\lambda(d_j)) \mid u \text{ is sampled first} \right],$$

## 23:10 Online Carpooling Using Expander Decompositions

is  $O(1)$  regardless of the choice of  $u$ , where we assume that  $i$  is the random vertex which is assigned  $-1$  by the  $(1 + \beta)$ -process, and  $j$  is the random vertex which is assigned  $+1$ . The proof of the lemma then follows by removing the conditioning on  $u$ .

Following [5, 4], we use the first two terms of the Taylor expansion of  $\cosh(\cdot)$  to upper bound the difference terms of the form  $\cosh(x + 1) - \cosh(x)$  and  $\cosh(x - 1) - \cosh(x)$ . To this end, note that, if  $|\epsilon| \leq 1$  and  $\lambda < 1$ , we have that

$$\begin{aligned} \cosh(\lambda(x + \epsilon)) - \cosh(\lambda x) &\leq \epsilon \lambda \sinh(\lambda x) + \frac{\epsilon^2}{2!} \lambda^2 \cosh(\lambda x) + \frac{\epsilon^3}{3!} \lambda^3 \sinh(\lambda x) + \dots \\ &\leq \epsilon \lambda \sinh(\lambda x) + \epsilon^2 \lambda^2 \cosh(\lambda x). \end{aligned}$$

Using this, we proceed to bound the following quantity (by setting  $\epsilon = -1$  and  $1$  respectively):

$$\mathbb{E}_{v \sim \mathbf{w}} \left[ \underbrace{-\lambda(\sinh(\lambda d_i) - \sinh(\lambda d_j))}_{=: -L} + \underbrace{\lambda^2(\cosh(\lambda d_i) + \cosh(\lambda d_j))}_{=: Q} \mid u \text{ is sampled first} \right].$$

We refer to  $L = \lambda(\sinh(\lambda d_i) - \sinh(\lambda d_j))$  and  $Q = \lambda^2(\cosh(\lambda d_i) + \cosh(\lambda d_j))$  as the *linear* and *quadratic* terms, since they arise from the first- and second-order derivatives in the Taylor expansion.

To further simplify our exposition, we define the following random variables:

- (i)  $u_>$  is the identity of the vertex among  $u, v$  with higher discrepancy, and  $u_<$  is the other vertex. Hence we have that  $d_{u_>} \geq d_{u_<}$ .
- (ii)  $G$  denotes the random variable  $\lambda(\sinh(\lambda d_{u_>}) - \sinh(\lambda d_{u_<}))$ , which indicates an analogous term to  $L$ , but if we exclusively did a greedy signing always (recall that the greedy algorithm would always decrease the larger discrepancy, but the  $(1 + \beta)$ -process follows a uniformly random signing with probability  $(1 - \beta)$  and follows the greedy rule only with probability  $\beta$ ).

Finally, for any vertex  $w \in V$ , we let  $\text{Danger}(w) = \{v : |d_w - d_v| < \frac{2}{\lambda}\}$  to denote the set of vertices with discrepancy close to that of  $w$ , where the gains from the term corresponding to  $\beta G$  are insufficient to compensate for the increase due to  $Q$ .

We are now ready to proceed with the proof. Firstly, note that, since the  $(1 + \beta)$ -process follows the greedy algorithm with probability  $\beta$  (independent of the choice of the sampled vertices  $u$  and  $v$ ), we have that

$$\mathbb{E}_v[L \mid u \text{ is sampled first}] = (1 - \beta)0 + \beta \mathbb{E}_v[G \mid u \text{ is sampled first}]. \quad (5)$$

Intuitively, the remainder of the proof proceeds as follows: suppose  $d_{u_>}$  and  $d_{u_<}$  are both non-negative (the intuition for the other cases are similar). Then,  $Q$  is proportional to  $\lambda^2 \cosh(\lambda d_{u_>})$ . Now, if  $d_{u_>} - d_{u_<}$  is sufficiently large, then  $G$  is proportional to  $\lambda \sinh(\lambda d_{u_>})$ , which in turn is close to  $\lambda \cosh(\lambda d_{u_>})$ . As a result, we get that as long as  $\lambda = O(\beta)$ , the term  $-\beta G + Q$  can be bounded by 0 for each choice of  $v$  such that  $d_{u_>} - d_{u_<}$  is large.

However, what happens when  $d_{u_>} - d_{u_<}$  is small, i.e., when  $v$  falls in  $\text{Danger}(u)$ ? Here, the  $Q$  term is proportional to  $\lambda^2 \cosh(\lambda d_u)$ , but the  $G$  term might be close to 0, and so we can't argue that  $-\beta G + Q \leq O(1)$  in these events. Hence, we resort to an amortized analysis by showing that (i) when  $v \notin \text{Danger}(u)$ ,  $-\beta G$  can not just compensate for  $Q$ , it can in fact compensate for  $\frac{1}{\sqrt{\lambda}} Q \geq \frac{1}{\sqrt{\lambda}} \cdot \lambda^2 \cosh(\lambda d_u)$ , and secondly, (ii) the probability over a random choice of  $v$  of  $v \notin \text{Danger}(u)$  is at least  $\sqrt{\lambda}$ , provided  $\Phi$  is bounded to begin with. The overall proof then follows from taking an average over all  $v$ .

Hence, in what follows, we will show that in expectation the magnitude of  $\beta G$  can compensate for a suitably large multiple of  $Q$  when  $v \notin \text{Danger}(u)$ .

▷ **Claim 15.** Let  $\beta \geq 6\lambda$ . For any fixed choice of vertices  $u$  and  $v$  such that  $v \notin \text{Danger}(u)$ , we have  $G := \lambda(\sinh(\lambda d_{u>}) - \sinh(\lambda d_{u<})) \geq \frac{\lambda}{3}(\cosh(\lambda d_u) + \cosh(\lambda d_v) - 4)$ .

*Proof.* The proof is a simple convexity argument. To this end, suppose both  $d_u, d_v \geq 0$ . Then since  $\sinh(x)$  is convex when  $x \geq 0$  and its derivative is  $\cosh(x)$ , we get that

$$\sinh(\lambda d_{u>}) - \sinh(\lambda d_{u<}) \geq \lambda \cosh(\lambda d_{u<}) \cdot |d_u - d_v| \geq 2 \cosh(\lambda d_{u<}),$$

using  $v \notin \text{Danger}(u)$ . But since  $|\sinh(x) - \cosh(x)| \leq 1$ , we get that

$$\sinh(\lambda d_{u>}) - \sinh(\lambda d_{u<}) \geq 2 \sinh(\lambda d_{u<}) - 2.$$

Therefore,  $\sinh(\lambda d_{u<}) \leq \frac{1}{3}(\sinh(\lambda d_{u>}) + 1)$ . Now substituting, and using the monotonicity of  $\sinh$  and its closeness to  $\cosh$ , we get  $G$  is at least

$$\frac{2\lambda}{3}(\sinh(\lambda d_{u>}) - 1) \geq \frac{\lambda}{3}(\sinh(\lambda d_{u>}) + \sinh(\lambda d_{u<}) - 2) \geq \frac{\lambda}{3}(\cosh(\lambda d_u) + \cosh(\lambda d_v) - 4).$$

The case of  $d_u, d_v \leq 0$  follows from setting  $d'_u = |d_u|, d'_v = |d_v|$  and using the above calculations, keeping in mind that  $\sinh$  is an odd function but  $\cosh$  is even. Finally, when  $d_{u<}$  is negative but  $d_{u>}$  is positive,

$$\begin{aligned} G &= \lambda(\sinh(\lambda d_{u>}) - \sinh(\lambda d_{u<})) = \lambda(\sinh(\lambda d_{u>}) + \sinh(\lambda |d_{u<}|)) \\ &\geq \frac{\lambda}{3}(\cosh(\lambda d_{u>}) + \cosh(\lambda d_{u<}) - 2) \geq \frac{\lambda}{3}(\cosh(\lambda d_u) + \cosh(\lambda d_v) - 4). \quad \blacktriangleleft \end{aligned}$$

▷ **Claim 16.** Let  $\beta \geq 6\lambda$ . For any fixed choice of vertices  $u$  and  $v$  such that  $v \notin \text{Danger}(u)$ , we have  $-\beta G + \left(1 + \frac{1}{\sqrt{\lambda}}\right) Q \leq O(1)$ .

*Proof.* Recall that  $G = \lambda(\sinh(\lambda d_{u>}) - \sinh(\lambda d_{u<}))$ . Now, let  $A$  denote  $\cosh(\lambda d_u) + \cosh(\lambda d_v)$ . Then, by definition of  $Q$  and from Claim 15, we have that

$$-\beta G + \left(1 + \frac{1}{\sqrt{\lambda}}\right) Q \leq -\frac{\beta\lambda}{3}(A-4) + \left(1 + \frac{1}{\sqrt{\lambda}}\right) \lambda^2 A \leq \frac{4\lambda\beta}{3} + \left(\lambda^2 + \lambda^{\frac{3}{2}} - \frac{\lambda\beta}{3}\right) A \leq \lambda\beta$$

is at most  $O(1)$ , assuming  $\beta \geq 6\lambda \geq 3(\lambda + \sqrt{\lambda})$ , and recalling that  $\lambda, \beta$  are at most 1.  $\blacktriangleleft$

We now proceed with our proof using two cases:

**Case (i):**  $|d_u| \leq \frac{10}{\lambda}$ . In this case, note that the  $Q$  term is

$$\begin{aligned} &\mathbb{E}_v[Q \mid u \text{ is sampled first}] \\ &= \mathbb{E}_v[Q \mid v \in \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \in \text{Danger}(u) \mid u \text{ is sampled first}] \\ &\quad + \mathbb{E}_v[Q \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}] \\ &\leq O(1) + \mathbb{E}_v[Q \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}]. \end{aligned}$$

Here the inequality uses  $v \in \text{Danger}(u)$  and  $|d_u| \leq \frac{10}{\lambda}$  to infer that both  $|d_u|$  and  $|d_v|$  are  $\leq \frac{12}{\lambda}$ . Hence the  $Q$  term in this scenario will simply be a constant.

Next we analyze the  $L$  term. For the following, we observe that the algorithm chooses a random  $\pm 1$  signing with probability  $(1 - \beta)$ , and chooses the greedy signing with probability  $\beta$ , and moreover, this choice is independent of the random choices of  $u$  and  $v$ . Hence, the expected  $L$  term conditioned on the algorithm choosing a random signing is simply 0, and the expected  $L$  term conditioned on the algorithm choosing the greedy signing is simply the term  $\mathbb{E}[G]$ . Hence, we can conclude that:

## 23:12 Online Carpooling Using Expander Decompositions

$$\begin{aligned}
& \mathbb{E}_v[-L \mid u \text{ is sampled first}] \\
&= \mathbb{E}_v[-L \mid v \in \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \in \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\quad + \mathbb{E}_v[-L \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\leq \mathbb{E}_v[-\beta G \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}].
\end{aligned}$$

Adding the inequalities and applying Claim 16, we get  $\mathbb{E}_v[-L + Q \mid u \text{ is sampled first}] \leq O(1)$ .

**Case (ii):**  $|d_u| > \frac{10}{\lambda}$ . We first prove two easy claims.

▷ **Claim 17.** Suppose  $v \in \text{Danger}(u)$ . Then  $\cosh(\lambda d_v) \leq 8 \cosh(\lambda d_u)$ .

Proof. Assume w.l.o.g. that  $d_u, d_v \geq 0$ . Also, assume that  $d_v \geq d_u$ , otherwise there is nothing to prove. Now  $d_v \leq d_u + \frac{2}{\lambda}$ . So  $\frac{\cosh(\lambda d_v)}{\cosh(\lambda d_u)} \leq \sup_x \frac{\cosh(x+2)}{\cosh(x)}$ . The supremum on the right happens when  $x \rightarrow \infty$ , and then the ratio approaches  $e^2 < 8$ . ◁

▷ **Claim 18.** For any discrepancy vector  $\mathbf{d}^t$  such that  $\Phi(\mathbf{d}^t) \leq O((nT)^{10})$ , and for any  $u$  such that  $|d_u| > \frac{10}{\lambda}$ , we have  $\Pr[v \notin \text{Danger}(u)] \geq 8\sqrt{\lambda}$ , as long as  $\lambda = O(\log^{-4} nT)$ .

Proof. We consider the case that  $d_u > \frac{10}{\lambda}$ ; the case were  $d_u < -\frac{10}{\lambda}$  is similar.

Assume for a contradiction that  $\Pr[v \in \text{Danger}(u)] \geq 1 - 8\sqrt{\lambda}$ , and so  $\Pr[v \notin \text{Danger}(u)] \leq 8\sqrt{\lambda}$ . We first show that the cardinality of the set  $|w \notin \text{Danger}(u)|$  is small. Indeed, this follows immediately from our assumption on the minimum weight of any vertex in the statement of Lemma 14 being at least  $\gamma/n$  times the total weight. So we have that for every  $w$ , the probability of sampling  $w$  in the  $(1 + \beta)$ -process is at least  $\pi_w \geq \gamma/n$ , implying that the total number of vertices not in  $\text{Danger}(u)$  must be at most  $\frac{8\sqrt{\lambda} \cdot n}{\gamma}$ . This also means that the total number of vertices in  $\text{Danger}(u) \geq \frac{n}{2}$  since  $\gamma \geq \lambda^{1/4} \geq 16\sqrt{\lambda}$  for sufficiently small  $\lambda$ .

Since  $d_u > \frac{10}{\lambda}$ , we get that any vertex  $v \in \text{Danger}(u)$  satisfies  $d_v \geq d_u - \frac{2}{\lambda} \geq \frac{8}{\lambda}$ . Moreover, since  $\sum_v d_v = 0$ , it must be that the negative discrepancies must in total compensate for the total sum of discrepancies of the vertices in  $\text{Danger}(u)$ . Hence, we have that  $\sum_{w: d_w < 0} |d_w| \geq \sum_{v \in \text{Danger}(u)} d_v \geq |\{v : v \in \text{Danger}(u)\}| \cdot \frac{8}{\lambda} \geq 0.5n \cdot \frac{8}{\lambda}$ .

From the last inequality, and since  $|\{w : d_w < 0\}| \leq |\{w : w \notin \text{Danger}(u)\}| \leq \frac{8\sqrt{\lambda}n}{\gamma}$ , we get that there exists a vertex  $\tilde{w}$  s.t  $d_{\tilde{w}} < 0$  and  $|d_{\tilde{w}}| \geq \frac{\gamma}{8\sqrt{\lambda}n} \cdot \frac{4n}{\lambda} = \frac{\gamma}{2\lambda^{3/2}}$ . But this implies  $\Phi(\mathbf{d}^t) \geq \cosh(\lambda d_{\tilde{w}}) \geq \cosh\left(\frac{\gamma}{2\sqrt{\lambda}}\right) > (nT)^{10}$ , using that  $\lambda = O(\log^{-4} nT)$  and that  $\gamma \geq \lambda^{1/4}$ . So we get a contradiction on the assumption that  $\Phi(\mathbf{d}^t) \leq (nT)^{10}$ . ◁

Returning to the proof for the case of  $|d_u| \geq \frac{10}{\lambda}$ , we get that

$$\begin{aligned}
& \mathbb{E}_v[Q \mid u \text{ is sampled first}] \\
&= \mathbb{E}_v[Q \mid v \in \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \in \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\quad + \mathbb{E}_v[Q \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\leq 8\lambda^2 \cosh(\lambda d_u) \\
&\quad + \mathbb{E}[Q \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}],
\end{aligned}$$

where the first term in inequality follows from Claim 17.

Next we analyze the  $L$  term similarly:

$$\begin{aligned}
& \mathbb{E}_v[-L \mid u \text{ is sampled first}] \\
&= \mathbb{E}_v[-L \mid v \in \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \in \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\quad + \mathbb{E}_v[-L \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}] \\
&\leq \mathbb{E}_v[-\beta G \mid v \notin \text{Danger}(u), u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}],
\end{aligned}$$

where the last inequality follows using the same arguments as in case (i). Adding these inequalities and applying Claim 16, we get that

$$\begin{aligned}
\mathbb{E}_v[-L + Q \mid u \text{ is sampled first}] &\leq O(1) + 8\lambda^2 \cosh(\lambda d_u) \\
&\quad - \frac{1}{\sqrt{\lambda}} \cdot \mathbb{E}_v[Q \mid u \text{ is sampled first}] \cdot \Pr[v \notin \text{Danger}(u) \mid u \text{ is sampled first}].
\end{aligned}$$

To complete the proof of Lemma 14, we note that  $Q \geq \lambda^2 \cosh(\lambda d_u)$ , and use Claim 18 to infer that  $\Pr[v \notin \text{Danger}(u)] \geq 8\sqrt{\lambda}$ . This implies

$$\mathbb{E}_v[-L + Q \mid u \text{ is sampled first}] \leq O(1) + 8\lambda^2 \cosh(\lambda d_u) - 8\lambda^2 \cosh(\lambda d_u) \leq O(1). \blacktriangleleft$$

We now can use this one-step expected potential change for the  $(1 + \beta)$ -process to get the following result for the original expander process:

**Proof of Theorem 6.** Combining Lemma 14 and Lemma 11, we get that in the expander process, if we condition on the random choices made until time  $t$ , if  $\Phi(\mathbf{d}^t) \leq (nT)^{10}$ , then  $\mathbb{E}[\Phi(\mathbf{d}^{t+1}) - \Phi(\mathbf{d}^t)] \leq C$  for some constant  $C$ . The potential starts off at  $n$ , so if it ever exceeds  $CT(nT)^5$  in  $T$  steps, there must be a time  $t$  such that  $\Phi(\mathbf{d}^t) \leq Ct(nT)^5$  and the increase is at least  $C(nT)^5$ . But the expected increase at this step is at most  $C$ , so by Markov's inequality the probability of increasing by  $C(nT)^5$  is at most  $1/(nT)^5$ . Now a union bound over all times  $t$  gives that the potential exceeds  $CT(nT)^5 \leq (nT)^{10}$  with probability at most  $T/(nT)^5 = 1/\text{poly}(nT)$ . But then  $\cosh(\lambda d_v^t) \leq (nT)^{10}$ , and therefore  $d_v^t \leq O(\lambda \log(nT)^{10}) = O(\log^3 nT)$  for all vertices  $v$  and time  $t$ .  $\blacktriangleleft$

In summary, if the underlying graph is  $\gamma$ -weakly-regular for  $\gamma \geq \Omega(\log^{-1} nT)$ , and has expansion  $\alpha \geq \Omega(\log^{-2} nT)$ , the greedy process maintains a poly-logarithmic discrepancy.

## 2.5 Putting It Together

We briefly describe the expander decomposition procedure and summarize the final algorithm.

► **Theorem 19** (Decomposition into Weakly-Regular Expanders). *Any graph  $G = (V, E)$  can be decomposed into an edge-disjoint union of smaller graphs  $G_1 \uplus G_2 \dots \uplus G_k$  such that each vertex appears in at most  $O(\log^2 n)$  many smaller graphs, and (b) each of the smaller subgraphs  $G_i$  is a  $\frac{\alpha}{4}$ -weakly regular  $\alpha$ -expander, where  $\alpha = O(1/\log n)$ .*

The proof is in the full version. So, given a graph  $G = (V, E)$ , we use Theorem 19 to partition the edges into a union of  $\frac{\alpha}{4}$ -weakly regular  $\alpha$ -expanders, namely  $H_1, \dots, H_s$ , where  $\alpha = O(1/\log n)$ . Further, each vertex in  $V$  appears in at most  $O(\log^2 n)$  of these expanders. For each graph  $H_i$ , we run the greedy algorithm independently. More formally, when an edge  $e$  arrives, it belongs to exactly one of the subgraphs  $H_i$ . We orient this edge with respect to the greedy algorithm running on  $H_i$ . Theorem 6 shows that the discrepancy of each vertex in  $H_i$  remains  $O(\log^5(nT))$  for each time  $t \in [0 \dots T]$  with high probability. Since each vertex in  $G$  appears in at most  $O(\log^2 n)$  such expanders, it follows that the discrepancy of any vertex in  $G$  remains  $O(\log^7 n + \log^5 T)$  with high probability. This proves Theorem 1.

---

**References**

---

- 1 Miklós Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J. Schulman, and Orli Waarts. Fairness in scheduling. *J. Algorithms*, 29(2):306–357, 1998.
- 2 Ryan Alweiss, Yang P. Liu, and Mehtaab Sawhney. Discrepancy minimization via a self-balancing walk. *CoRR*, abs/2006.14009, 2020.
- 3 Nikhil Bansal, Haotian Jiang, Raghu Meka, Sahil Singla, and Makrand Sinha. Online discrepancy minimization for stochastic arrivals. *CoRR*, abs/2007.10622, 2020.
- 4 Nikhil Bansal, Haotian Jiang, Sahil Singla, and Makrand Sinha. Online vector balancing and geometric discrepancy. In *Proceedings of STOC*, 2020.
- 5 Nikhil Bansal and Joel H. Spencer. On-line balancing of random inputs. *CoRR*, abs/1903.06898, 2019. URL: <http://arxiv.org/abs/1903.06898>.
- 6 Imre Bárány. On a Class of Balancing Games. *J. Comb. Theory, Ser. A*, 26(2):115–126, 1979.
- 7 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *CoRR*, abs/2004.08432, 2020. [arXiv:2004.08432](https://arxiv.org/abs/2004.08432).
- 8 Bernard Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, 2001.
- 9 Raaz Dwivedi, Ohad N. Feldheim, Ori Gurel-Gurevich, and Aaditya Ramdas. The power of online thinning in reducing discrepancy. *Probability Theory and Related Fields*, 174:103–131, 2019.
- 10 G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 2 edition, 1952.
- 11 Haotian Jiang, Janardhan Kulkarni, and Sahil Singla. Online geometric discrepancy for stochastic arrivals with applications to envy minimization. *CoRR*, abs/1910.01073, 2019.
- 12 Jiri Matousek. *Geometric discrepancy: An illustrated guide*, volume 18. Springer Science & Business Media, 2009.
- 13 Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the  $(1 + \beta)$ -choice process. *Random Struct. Algorithms*, 47(4):760–775, 2015.
- 14 Joel Spencer. Balancing games. *J. Comb. Theory, Ser. B*, 23(1):68–74, 1977.

# On the (Parameterized) Complexity of Almost Stable Marriage

**Sushmita Gupta**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
sushmitagupta@imsc.res.in

**Pallavi Jain**

Indian Institute of Technology Jodhpur, India  
pallavi@iitj.ac.in

**Sanjukta Roy**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
sanjukta@imsc.res.in

**Saket Saurabh**

The Institute of Mathematical Sciences, HBNI, Chennai, India  
University of Bergen, Norway  
saket@imsc.res.in

**Meirav Zehavi**

Ben Gurion University of the Negev, Beer Sheva, Israel  
meiravze@bgu.ac.il

---

## Abstract

In the STABLE MARRIAGE problem, when the preference lists are complete, all agents of the smaller side can be matched. However, this need not be true when preference lists are incomplete. In most real-life situations, where agents participate in the matching market voluntarily and submit their preferences, it is natural to assume that each agent wants to be matched to someone in his/her preference list as opposed to being unmatched. In light of the Rural Hospital Theorem, we have to relax the “no blocking pair” condition for stable matchings in order to match more agents. In this paper, we study the question of matching more agents with fewest possible blocking edges. In particular, the goal is to find a matching whose size exceeds that of a stable matching in the graph by at least  $t$  and has at most  $k$  blocking edges. We study this question in the realm of parameterized complexity with respect to several natural parameters,  $k, t, d$ , where  $d$  is the maximum length of a preference list. Unfortunately, the problem remains intractable even for the combined parameter  $k + t + d$ . Thus, we extend our study to the local search variant of this problem, in which we search for a matching that not only fulfills each of the above conditions but is “closest”, in terms of its symmetric difference to the given stable matching, and obtain an FPT algorithm.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Stable Matching, Parameterized Complexity, Local Search

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.24

**Related Version** A full version of the paper is available at <https://arxiv.org/pdf/2005.08150.pdf>.

**Funding** *Sushmita Gupta*: was supported by SERB-Starting Research Grant (SRG/2019/001870).

*Saket Saurabh*: Received funding from European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

*Meirav Zehavi*: was supported by ISF grant (no.1176/18) and BSF grant no. 2018302).



© Sushmita Gupta, Pallavi Jain, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 24; pp. 24:1–24:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## 1 Introduction

Matching various entities to available resources is of great practical importance, exemplified in matching college applicants to college seats, medical residents to hospitals, preschoolers to kindergartens, unemployed workers to jobs, organ donors to recipients, and so on [2, 14, 19, 21]. It is noteworthy that in the applications mentioned above, it is not enough to merely match an entity to any of the available resources. It is imperative, in fact, mission-critical, to create matches that fulfil some predefined notions of compatibility, suitability, acceptability, and so on. Gale and Shapley introduced the fundamental theoretical framework to study such two-sided matching markets in the 1960s. They envisioned a matching outcome as a *marriage* between the members of the two sides, and a desirable outcome representing a *stable marriage*. The algorithm proffered by them has since attained wide-scale recognition as the Gale-Shapley stable marriage/matching algorithm [14]. Stability is one of the acceptability criteria for matching in which an unmatched pair of agent should not prefer each other over their matched partner.

Of the many characteristic features of the two-sided matching markets, there are certain aspects that stand out and are supported by both theoretical and empirical evidence – particularly notable is the curious aspect that for a given market with strict preferences on both sides,<sup>1</sup> no matter what the stable matching outcome is, the specific *number* of resources matched on either side always remains the same. This fact encapsulated by The Rural Hospital’s Theorem [30, 31] states that no matter what stable matching algorithm is deployed, the exact *set* (rather than only the number) of resources that are matched on either side is the same. In other words, *there is a trade-off between size and stability such that any increase in size must be paid for by sacrificing stability*. Indeed, it is not hard to find instances in which as much as half of the available resources are unmatched in every stable matching. Such gross underutilization of critical and potentially expensive resources has not gone unaddressed by researchers. In light of The Rural Hospital Theorem, many variations have been considered, some important ones being: enforcing lower and upper capacities, forcing some matches, forbidding some matches, relaxing the notion of stability, and finally foregoing stability altogether in favor of size [2, 3, 7, 16, 22, 34].

We formalize the trade-off mentioned above between size and stability in terms of the ALMOST STABLE MARRIAGE problem. The classical STABLE MARRIAGE problem takes as an instance a bipartite graph  $G = (A \cup B, E)$ , where  $A$  and  $B$  denote the set of vertices representing the agents on the two sides and  $E$  denotes the set of edges representing acceptable matches between vertices on different sides, and a preference list of every vertex in  $G$  over its neighbors. Thus, the length of the preference list of a vertex is the same as its degree in the graph. A *matching* is defined as a subset of the set of edges  $E$  such that no vertex appears in more than one edge in the matching. An edge in a matching represents a match such that the endpoints of a matching edge are said to be the *matching partners* of each other, and an unmatched vertex is deemed to be self-matched. A matching  $\mu$  is said to be *stable* in  $G$  if there does not exist a *blocking edge* with respect to  $\mu$ , defined to be an edge  $e \in E \setminus \mu$  whose endpoints rank each other higher (in their respective preference lists) than their matching partners in  $\mu$ .<sup>2</sup> The goal of the STABLE MARRIAGE problem is to find a stable matching. We define the ALMOST STABLE MARRIAGE problem as follows.

<sup>1</sup> In most real-life applications, it is unreasonable if not unrealistic to expect each of the agents to rank all the agents on the other side. That is, the graph  $G$  is highly unlikely to be complete.

<sup>2</sup> Every candidate is assumed to prefer being matched to any of its neighbors to being self-matched.

**ALMOST STABLE MARRIAGE (ASM)**

**Input:** A bipartite graph  $G = (A \cup B, E)$ , a set  $\mathcal{L}$  containing the preference list of each vertex, and non-negative integers  $k$  and  $t$ .

**Question:** Does there exist a matching whose size is at least  $t$  more than the size of a stable matching in  $G$  such that the matching has at most  $k$  blocking edges?

In ASM, we hope for a matching that is larger than a stable matching but may contain some blocking edges. The above problem quantifies these two variables:  $t$  and  $k$  denote the minimum increase in the size and the allowable number of blocking edges, respectively.

We note that Biró et al. [3] considered the problem of finding, among all matchings of maximum size, one that has the fewest blocking edges, and showed the NP-hardness of the problem even when the degree of the graph is at most three. Since one can find a maximum matching and a stable matching in the given graph in polynomial time [27, 14], their NP-hardness result implies NP-hardness for ASM even when the degree is at most three by setting  $t$  to be the difference between the size of a maximum matching and the size of a stable matching.

**Our Contribution and Methods.** We study the parameterized complexity of ASM with respect to parameters  $k$  and  $t$ ; a combination that is not settled by Biró et al. [3]. Our first result exhibits a strong guarantee of intractability. We exhibit parameterized intractability of ASM in a very restrictive setting where the degree of the given graph is three.

► **Theorem 1.** *ASM is W[1]-hard with respect to  $k + t$ , even when the maximum degree of the given graph is at most three.*

We prove Theorem 1 by showing a polynomial-time many-to-one parameter preserving reduction from the MULTICOLORED CLIQUE (MCQ, in short) problem to ASM. In the MULTICOLORED CLIQUE problem, given a graph  $G = (V, E)$  and a partition of  $V(G)$  into  $k$  parts, say  $V_1, \dots, V_k$ ; the goal is to decide the existence of a set  $S \subseteq V(G)$  such that  $|S \cap V_i| = 1$ , for all  $i \in [k]$ , and  $G[S]$  induces a clique, that is, there is an edge between every pair of vertices in  $G[S]$ . MCQ is known to be W[1]-hard [29, 12] with respect to  $k$ .

In light of the intractability result in Theorem 1, we are hard pressed to recalibrate our expectations of what is algorithmically feasible in an efficient manner. Therefore, we consider a local search approach for this problem, in which, instead of finding any matching whose size is at least  $t$  larger than the size of stable matching, we also want this matching to be “closest” in terms of its symmetric difference, to a stable matching. Such framework of local search has also been studied for other variants of the STABLE MARRIAGE problem by Marx and Schlotter [26, 25]. *We would like to emphasize that the notion of local search used here is different from the classical notion of local search heuristics/algorithms commonly used in practice [33].* We use the notion of local search that is well-defined and widely used in the domain of parameterized complexity, as exemplified by Marx and Schlotter [26, 25], and has also been applied to study several other optimization problems [11, 18, 20, 23, 24, 25, 32]. The question is formally defined as follows.

**LOCAL SEARCH-ASM (LS-ASM)**

**Input:** A bipartite graph  $G = (A \cup B, E)$ , a set  $\mathcal{L}$  containing the preference list of every vertex, a stable matching  $\mu$ , and non-negative integers  $k$ ,  $q$ , and  $t$ .

**Question:** Does there exist a matching  $\eta$  of size at least  $|\mu| + t$  with at most  $k$  blocking edges such that the symmetric difference between  $\mu$  and  $\eta$  is at most  $q$ ?

## 24:4 On the (Parameterized) Complexity of Almost Stable Marriage

Unsurprisingly perhaps, the existence of a stable matching in the proximity of which we wish to find a solution does not readily mitigate the computational hardness of the problem, as evidenced by Theorem 2. This result is a consequence of the properties of the reduction used in the proof of Theorem 1. The NP-hardness of LS-ASM also follows from the NP-hardness of ASM as we can set  $q$  to be  $2n$ , the maximum possible size of  $\mu \cup \eta$ .

► **Theorem 2** (♣).<sup>3</sup> *LS-ASM is  $W[1]$ -hard with respect to  $k + t$ , even when the maximum degree of the given graph is at most three.*

In our quest for a parameterization that makes the problem tractable, we investigate LS-ASM with respect to  $k + q + t$ .

► **Theorem 3** (♣). *LS-ASM is  $W[1]$ -hard with respect to  $k + q + t$ .*

To prove Theorem 3, we give a polynomial-time many-to-one parameter preserving reduction from MCQ to LS-ASM. In the instance constructed to prove Theorem 1,  $q$  is not a function of  $k$ . We mimic the idea of gadget construction in that proof and ensure that  $q$  is a function of  $k$ . However, in this effort, the degree of the graph increases. Consequently, the result in Theorem 3 does not hold for constant degree graphs or even when the degree is a function of  $k$ . This trade-off between  $q$  and the degree of the graph in the instances that establish intractability is not a coincidence, as implied by our next result.

► **Theorem 4.** *There exists an algorithm that, given an instance of LS-ASM, solves the instance in  $2^{\mathcal{O}(q \log d) + o(dq)} n^{\mathcal{O}(1)}$  time, where  $n$  is the number of vertices in the given graph, and  $d$  is the maximum degree of the given graph.*

To prove Theorem 4, we begin by using the technique of *random separation* based on color coding, in which the underlying idea is to highlight the solution that we are looking for with high probability. Suppose that  $\eta$  is a hypothetical solution to the given instance of LS-ASM. Note that to find the matching  $\eta$ , it is enough to find the edges that are in the symmetric difference of  $\mu$  and  $\eta$ , denoted by  $\mu \Delta \eta$ . Thus, using the technique of random separation, we wish to highlight the edges in  $\mu \Delta \eta$ . We achieve this goal using two layers of randomization. The first one separates vertices that appear in  $\mu \Delta \eta$ , denoted by the set  $V(\mu \Delta \eta)$ , from its neighbors, by independently coloring vertices 1 or 2. Let the vertices appearing in  $V(\mu \Delta \eta)$  be colored 1 and its neighbors that are not in  $V(\mu \Delta \eta)$  be colored 2. Observe that the matching partner of the vertices which are not in  $V(\mu \Delta \eta)$  is the same in both  $\mu$  and  $\eta$ . Therefore, we search for a solution locally in vertices that are colored 1. Let  $G_1$  be the graph induced on the vertices that are colored 1. At this stage we use a second layer of randomization on edges of  $G_1$ , and independently color each edge with 1 or 2. This separates edges that belong to  $\mu \Delta \eta$  (say colored 1) from those that do not belong to  $\mu \Delta \eta$ . Now for each component of  $G_1$ , we look at the edges that have been colored 1, and compute the number of blocking edges, the increase in size and increase in the symmetric difference, if we modify using the  $\mu$ -alternating paths/cycle that are present in this component. This leads to an instance of the TWO-DIMENSIONAL KNAPSACK (2D-KP) problem, which we solve in polynomial time using a known pseudo-polynomial time algorithm for 2D-KP [17]. We derandomize this algorithm using the notion of an  $n$ - $p$ - $q$ -lopsided universal family [13]. Table 1 summarizes the results for ASM and LS-ASM.

---

<sup>3</sup> Proofs marked by [♣] are deferred to the full version of the paper.

■ **Table 1** Summary of the results for ASM and LS-ASM. Results in blue row are implied from Theorem 7 in [3].

ASM	LS-ASM
NP-hard for $d = 3$ [3]	NP-hard for $d = 3$ [3]
W[1]-hard for $d = 3$ wrt $k + t$ [Thm. 1]	W[1]-hard wrt $k + q + t$ [Thm. 3] FPT wrt $q + d$ [Thm. 4]

**Related Work.** We present here some variants of the STABLE MARRIAGE problem which are closely related to our model. In the past, the notion of “almost stability” is defined for the STABLE ROOMMATE problem [1]. In the STABLE ROOMMATE problem, the goal is to find a stable matching in an arbitrary graph. As opposed to STABLE MARRIAGE, in which the graph is a bipartite graph, an instance of STABLE ROOMMATE might not admit a stable matching. Therefore, the notion of almost stability is defined for the STABLE ROOMMATE problem, in which the goal is to find a matching with a minimum number of blocking edges. This problem is known as the ALMOST STABLE ROOMMATE problem. Abraham et al. [1] proved that the ALMOST STABLE ROOMMATE problem is NP-hard. Biro et al. [4] proved that the problem remains NP-hard even for constant-sized preference lists and studied it in the realm of approximation algorithms. Chen et al. [5] studied this problem in the realm of parameterized complexity and showed that the problem is W[1]-hard with respect to the number of blocking edges even when the maximum length of every preference list is five.

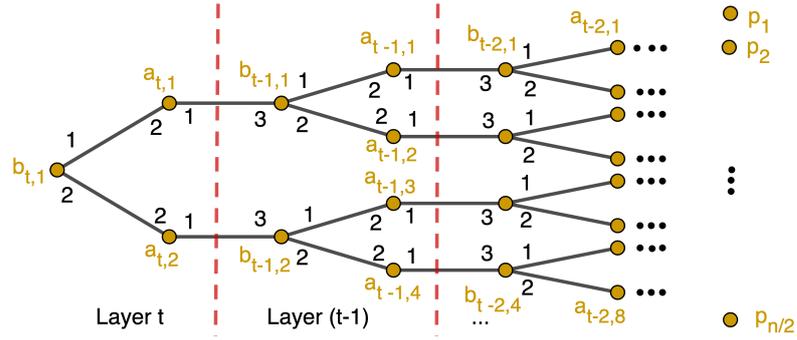
Later in 2010, Biró et al. [3] considered the problem of finding, among all matchings of the maximum size, one that has the fewest blocking edges, in a bipartite graph and showed that the problem is NP-hard and not approximable within  $n^{1-\epsilon}$ , for any  $\epsilon > 0$  unless P=NP.

The problem of finding the maximum sized stable matching in the presence of ties and incomplete preference lists, maxSMTI, has striking resemblance with ASM. In maxSMTI, the decision of resolving each tie comes down to deciding who should be at the top of each of tied lists, mirrors the choice we have to make in ASM in rematching the vertices who will be part of a blocking edge in the new matching. Despite this similarity, the W[1]-hardness result presented in [26, Theorem 7] does not yield the hardness result of ASM and LS-ASM as the reduction is not likely to be parameteric in terms of  $k + t$  and  $k + t + q$ , or have the degree bounded by a constant. For other variants of the STABLE MARRIAGE problem, we refer the reader to [6, 21, 15, 19].

## 2 Preliminaries

*Sets.* We denote the set of natural numbers  $\{1, \dots, \ell\}$  by  $[\ell]$ . For two sets  $X$  and  $Y$ , we use notation  $X \triangle Y$  to denote the symmetric difference between  $X$  and  $Y$ . For any ordered set  $X$ , and an appropriately defined value  $t$ ,  $X(t)$  denotes the  $t^{\text{th}}$  element of the set  $X$ . Conversely, suppose that  $x$  is  $t^{\text{th}}$  element of the set  $X$ , then  $\sigma(X, x) = t$ .

**Graphs.** Let  $G$  be an undirected graph. We denote an edge between  $u$  and  $v$  as  $uv$ . The *neighborhood* of a vertex  $v$ , denoted by  $N_G(v)$ , is the set of all vertices adjacent to it. Analogously, the *(open) neighborhood* of a subset  $S \subseteq V$ , denoted by  $N_G(S)$ , is the set of vertices outside  $S$  that are adjacent to some vertex in  $S$ . A *component* of  $G$  is a maximal subgraph in which any two vertices are connected by a path. Let  $H$  be a subgraph of  $G$ . For a component  $C$  in  $H$ , we set  $N_G(C) = N_G(V(C))$ . The subscript may be omitted if the graph under consideration is clear from the context.



■ **Figure 1** Depiction of the top three layers of special vertices associated with the vertices of  $G'$ , as explained on page 7, where  $t = \log_2(n/2)$ . Yellow labels denote vertex labels and numbers on edges denote preferences. Analogously, we can depict the layers of the special vertices associated with the edges of  $G$ .

In the preference list of a vertex  $u$ , if  $v$  appears before  $w$ , then we say that  $u$  prefers  $v$  to  $w$ , and denote it as  $v \succ_u w$ . We call an edge in the graph a *static edge* if its endpoints prefer each other over any other vertex in the graph. For a matching  $\mu$ ,  $V(\mu) = \{u, v : uv \in \mu\}$ . If an edge  $uv \in \mu$ , then  $\mu(u) = v$  and  $\mu(v) = u$ . A vertex is called *saturated* in a matching  $\mu$ , if it is an endpoint of one of the edges in the matching  $\mu$ , otherwise it is an *unsaturated* vertex in  $\mu$ . If  $u$  is an unsaturated vertex in a matching  $\mu$ , then we write  $\mu(u) = \emptyset$ . For a matching  $\mu$  in  $G$ , a  $\mu$ -alternating path (cycle) is a path (cycle) whose edges alternate between matching edges of  $\mu$  and non-matching edges. A  $\mu$ -augmenting path is a  $\mu$ -alternating path that starts and ends at an unmatched vertex in  $\mu$ .

Unless specified, we will be using all general graph terminologies from the book of Diestel [9]. For parameterized complexity related definitions, we refer the reader to [8, 10, 28].

We conclude this section with a result that is used extensively in our analysis.

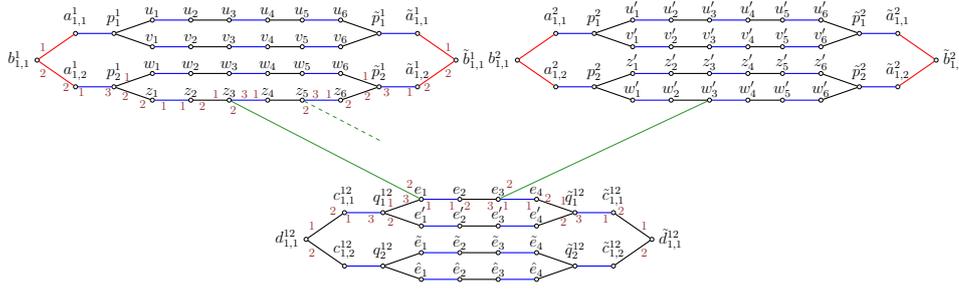
► **Proposition 1 (♣).** *Let  $\mu$  and  $\mu'$  denote two matchings in  $G$  such that  $\mu$  is stable and  $\mu'$  is not. Then, for each blocking edge with respect to  $\mu'$  we know that at least one of the endpoints has different matching partners in  $\mu$  and  $\mu'$ .*

### 3 W[1]-hardness of ASM

We give a polynomial-time parameter preserving many-to-one reduction from the W[1]-hard problem MULTICOLORED CLIQUE (MCQ) [29, 12] in which we are given a regular graph  $G = (V, E)$  and a partition of  $V(G)$  into  $k$  parts,  $V_1, \dots, V_k$ , and the objective is to decide if there exists a subset  $S \subseteq V(G)$  such that  $|S \cap V_i| = 1$ , for each  $i \in [k]$ , and the induced subgraph  $G[S]$  is a clique. Given an instance  $\mathcal{I} = (G, (V_1, \dots, V_k))$  of MCQ, we will next describe the construction of an instance  $\mathcal{J} = (G', \mathcal{L}, k', t)$  of ASM.

**Construction.** We begin by introducing some notations. For any  $\{i, j\} \subseteq [k]$ , such that  $i < j$ , we use  $E_{ij}$  to denote the set of edges between sets  $V_i$  and  $V_j$ . For each  $i \in [k]$ , we have  $|V_i| = n = 2^p$ , and for each  $\{i, j\} \subseteq [k]$ , we have  $|E_{ij}| = m = 2^{p'}$ , for some positive integers  $p$  and  $p'$  greater than one.<sup>4</sup> We assume that sets  $V_i$  (for each  $i \in [k]$ ) and  $E_{ij}$  (for

<sup>4</sup> Let  $m'$  be the maximum number of edges in any  $E_{ij}$ , where  $\{i, j\} \subseteq [k]$ . Let  $p'$  be the smallest positive integer greater than one such that  $m' \leq 2^{p'}$ . Then, for every  $\{i, j\} \subseteq [k]$ , add  $2^{p'} - |E_{ij}|$  isolated edges



■ **Figure 2** An illustration of construction of graph  $G'$  in the proof of  $W[1]$ -hardness of ASM for constant sized preference list. Here, blue colored edges belong to the stable matching  $\mu$ . Here,  $n = 4$ ,  $m = 4$ , and  $r_u = 2$ , for all  $u \in V(G)$ .

each  $\{i, j\} \subseteq [k], i < j$  have a canonical order, and thus for an appropriately defined value  $t$ ,  $V_i(t)$  ( $E_{ij}(t)$ ) and  $\sigma(V_i, v)$  ( $\sigma(E_{ij}, e)$ ), where  $v \in V_i$  and  $e \in E_{ij}$ , are uniquely defined. For each vertex  $u \in V(G)$ , let  $r_u$  denotes the degree of  $u$  in the graph  $G$ .

For each  $j \in [\log_2(m/2)]$ , let  $\beta_j = n/2^j$ , and  $\gamma_j = \beta_j/2$ . For each  $j \in [\log_2(m/2)]$ , let  $\rho_j = m/2^j$ , and  $\tau_j = \rho_j/2$ . Next, we are ready to describe the construction of the graph  $G'$ .

**Base vertices.**

- For each vertex  $u \in V(G)$ , we add  $2r_u + 2$  vertices in  $G'$ , denoted by  $\{u_i : i \in [2r_u + 2]\}$ , connected via a path:  $(u_1, \dots, u_{2r_u+2})$ .
- For each edge  $e \in E(G)$ , we have four vertices in  $G'$ , denoted by  $\{e_i : i \in [4]\}$ , connected via a path:  $(e_1, e_2, e_3, e_4)$ .

For each vertex  $u \in V(G)$ , we define a set  $\mathcal{E}_u \subseteq V(G')$  as follows. Let  $u \in V_i$ , for some  $i \in [k]$ . Then, for any edge  $e(= uv) \in E_{ij}$ , where  $j \in [k], j > i$ , we have that the vertex  $e_1 \in \mathcal{E}_u$ ; and for any edge  $e(= uv) \in E_{ji}$ , where  $j \in [k], j < i$ , we have that the vertex  $e_3 \in \mathcal{E}_u$ . Formally,

$$\mathcal{E}_u = \{e_1 \in V(G') : e = uv \in E_{ij}\} \cup \{e_3 \in V(G') : e = uv \in E_{ji}\}$$

We assume that the set  $\mathcal{E}_u$  has a canonical ordering. We encode the vertex-edge incidence relation in the graph  $G$  as follows: For each vertex  $u \in V(G)$  and value  $h \in [r_u]$ , the vertex  $u_{2h+1}$  in  $G'$  is a neighbor of the vertex  $\mathcal{E}_u(h)$ . Thus, the fact that the edge  $e$  is incident to a vertex  $u$  in  $G$ , is captured by the fact that a ‘‘copy’’ of  $e$  (namely  $e_1$  or  $e_3$ ) is adjacent to a ‘‘copy’’ of  $u$  in  $G'$ .

**Special vertices.** For each  $i \in [k]$ , we create the following special vertices associated with the vertices in  $V_i$ .

- For each  $\ell \in [\beta_1]$ , we add vertices  $p_\ell^i$  and  $\tilde{p}_\ell^i$  in  $V(G')$ . Let  $u$  and  $v$  denote the  $2\ell - 1^{st}$  and the  $2\ell^{th}$  vertices in  $V_i$ , respectively. Then, the vertex  $p_\ell^i$  is a neighbor of vertices  $u_1$  and  $v_1$ ; and the vertex  $\tilde{p}_\ell^i$  is a neighbor of vertices  $u_{2r_u+2}$  and  $v_{2r_v+2}$  in  $G'$ .

---

(an edge whose endpoints are of degree exactly one) to  $E_{ij}$ . Similarly, let  $n'$  be the maximum number of vertices in any  $V_i$ , where  $i \in [k]$ . Let  $p$  be the smallest positive integer greater than one such that  $n' \leq 2^p$ . Then, for every  $i \in [k]$ , add  $2^p - |V_i|$  isolated vertices to  $V_i$ . Note that if  $(G, (V_1, \dots, V_k))$  was a  $W[1]$ -hard instance of MCQ earlier, then so is the modified instance.

- For each  $j \in [\log_2(n/2)]$ , we add vertices in  $G'$  in *layers*, where the value of  $j$  gives the layer. Vertices in layer  $j$  are  $\{b_{j,\ell}^i : \ell \in [\beta_j/2]\} \cup \{a_{j,\ell}^i : \ell \in [\beta_j]\}$ . In the 1<sup>st</sup> layer,  $a_{1,\ell}^i$  is a neighbor of  $p_\ell^i$ . In the top layer, i.e.,  $j = \log_2(n/2)$ ,  $b_{j,1}^i$  is a neighbor of  $a_{j,1}^i$  and  $a_{j,2}^i$ . In intermediate layers, i.e.,  $1 < j < \log_2(n/2)$ , vertex  $b_{j,\ell}^i$  is adjacent to two vertices in its layer, namely  $a_{j,2\ell-1}^i, a_{j,2\ell}^i$  as well as one vertex from layer  $j + 1$ , namely  $a_{j+1,\ell}^i$ . Refer to Figure (1) for a depiction of two layers. Symmetrically, we define vertices  $\{\tilde{b}_{j,\ell}^i : \ell \in [\beta_j/2]\} \cup \{\tilde{a}_{j,\ell}^i : \ell \in [\beta_j]\}$  and define similar adjacencies for them as well; details are in Table 2.

For each  $\{i, j\} \subseteq [k]$ , where  $i < j$ , we create the following special vertices associated with the edges in  $E_{ij}$ .

- For each  $\ell \in [\rho_1]$ , we add vertices  $q_\ell^{ij}$  and  $\tilde{q}_\ell^{ij}$  to  $V(G')$ . Moreover, let  $e$  and  $e'$  denote the  $2\ell - 1$ <sup>st</sup> and  $2\ell$ <sup>th</sup> elements of  $E_{ij}$ , respectively. Then,  $q_\ell^{ij}$  is a neighbor of  $e_1$  and  $e'_1$ ; and symmetrically  $\tilde{q}_\ell^{ij}$  is a neighbor of  $e_4$  and  $e'_4$  in  $G'$ .
- As before, for each  $h \in [\log_2(m/2)]$ , we add vertices in  $G'$  in layers, where the value of  $h$  indicates the layer. Vertices in layer  $h$  are  $\{c_{h,\ell}^{ij} : \ell \in [\rho_j/2]\} \cup \{d_{h,\ell}^{ij} : \ell \in [\rho_j]\}$ . In the 1<sup>st</sup> layer, vertex  $c_{1,\ell}^{ij}$  is a neighbor of  $q_\ell^{ij}$ . In the top layer, i.e.,  $h = \log_2(m/2)$ , vertex  $d_{h,1}^{ij}$  is a neighbor of  $c_{h,1}^{ij}$  and  $c_{h,2}^{ij}$ . In intermediate layers, i.e.,  $1 < h < \log_2(m/2)$ , vertex  $d_{h,\ell}^{ij}$  is adjacent to two vertices in its layer, namely  $c_{h,2\ell-1}^{ij}, c_{h,2\ell}^{ij}$  as well as one vertex from layer  $h + 1$ , namely  $c_{h+1,\ell}^{ij}$ . Symmetrically, we define vertices  $\{\tilde{c}_{h,\ell}^{ij} : \ell \in [\rho_j/2]\} \cup \{\tilde{d}_{h,\ell}^{ij} : \ell \in [\rho_j]\}$  and define similar adjacencies for these vertices; details are in Table 2.

Figure 2 illustrates the construction of  $G'$ . The preference list of each vertex in  $G'$  is presented in Table 2.

**Parameter:** We set  $k' = k^2$ , and  $t = k + k(k-1)/2$ . This completes the construction of an instance of ASM. Clearly, this construction can be carried out in polynomial time. Since  $|V_i| = n$  and  $|E_{ij}| = m$ , for every  $\{i, j\} \subseteq [k]$ , we have  $2nk + 4mk(k-1)$  many base vertices and  $4nk + 2mk(k-1) - 3k - 3k^2$  many special vertices. Thus, in total we have

$$|V(G')| = 6mk(k-1) + 6nk - 3k - 3k^2. \quad (\text{I})$$

The rest of the proof of Theorem 1 is deferred to the full version of the paper.

## 4 FPT Algorithm for LS-ASM

In this section, we give an FPT algorithm for LS-ASM with respect to  $q + d$  (Theorem 4). Recall that  $d$  is the degree of the graph  $G$ , and  $q$  is the symmetric difference between a solution matching and the given stable matching  $\mu$ . Before presenting our algorithm, we prove that there exists a solution,  $\gamma$ , to  $(G, \mathcal{L}, \mu, k, q, t)$  such that in every component of  $G[V(\mu \Delta \gamma)]$ , the number of  $\gamma$ -edges (edges that are in  $\gamma$ ) is more than the number of  $\mu$ -edges in this component. We will need such a solution for a technical purpose which will be cleared later in Phase III of the algorithm.

► **Lemma 5.** *There exists a solution  $\gamma$  to  $(G, \mathcal{L}, \mu, k, q, t)$  such that for every component  $C$  of  $G[V(\mu \Delta \gamma)]$ ,  $|E(C) \cap \gamma| > |E(C) \cap \mu|$ .*

The proof of Lemma 5 follows by starting with a solution  $\gamma$  and then replacing the edges in  $\mu$  with the edges in  $\gamma$  only in those components of  $G[V(\mu \Delta \gamma)]$ , where  $|\gamma| > |\mu|$ .

We begin with the description of a randomized algorithm which will be derandomized later using  $n$ - $p$ - $q$ -lopsided universal family [13]. Our algorithm has three phases: Vertex Separation, Edge Separation, and Size-Fitting. Given an instance  $(G, \mathcal{L}, \mu, k, q, t)$  of LS-ASM, we proceed as follows.

**Phase I: Vertex Separation.** We start with the following assumption.

■ **Table 2** Preference lists in the proof of Theorem 1. Here, for a set  $S$ , the notation  $\langle S \rangle$  denotes the order of preference over the vertices in this set.

For each vertex  $u \in V_i$ , where  $i \in [k]$ , we have the following preferences:

$$\begin{array}{ll} u_1: & \langle u_2, \tilde{p}_{\lceil \ell/2 \rceil}^i \rangle & \text{where for some } \ell \in [n], u = V_i(\ell). \\ u_{2h+1}: & \langle u_{2h}, \mathcal{E}_u(h), u_{2h+2} \rangle & \text{where } h \in [r_u] \\ u_{2h}: & \langle u_{2h-1}, u_{2h+1} \rangle & \text{where } h \in [r_u] \\ u_{2r_u+2}: & \langle u_{2r_u+1}, \tilde{p}_{\lceil \ell/2 \rceil}^i \rangle & \text{where for some } \ell \in [n], u = V_i(\ell). \end{array}$$

For the special vertices associated with  $V_i$ , we have the following preferences:

$$\begin{array}{ll} p_\ell^i: & \langle u_1, v_1, a_{1,\ell}^i \rangle & \text{where } \ell \in [n/2], u = V_i(2\ell - 1) \text{ and } v = V_i(2\ell) \\ \tilde{p}_\ell^i: & \langle u_{2r_u+2}, v_{2r_v+2}, \tilde{a}_{1,\ell}^i \rangle & \text{where } \ell \in [n/2], u = V_i(2\ell - 1) \text{ and } v = V_i(2\ell) \\ a_{1,\ell}^i: & \langle p_\ell^i, b_{1,\lceil \ell/2 \rceil}^i \rangle & \text{where } \ell \in [n/2] \\ \tilde{a}_{1,\ell}^i: & \langle \tilde{p}_\ell^i, \tilde{b}_{1,\lceil \ell/2 \rceil}^i \rangle & \text{where } \ell \in [n/2] \\ a_{j,\ell}^i: & \langle b_{j-1,\ell}^i, b_{j,\lceil \ell/2 \rceil}^i \rangle & \text{where } j \in [\log_2(n/2)] \setminus \{1\} \text{ and } \ell \in [n/2^j] \\ \tilde{a}_{j,\ell}^i: & \langle \tilde{b}_{j-1,\ell}^i, \tilde{b}_{j,\lceil \ell/2 \rceil}^i \rangle & \text{where } j \in [\log_2(n/2)] \setminus \{1\} \text{ and } \ell \in [n/2^j] \\ b_{j,\ell}^i: & \langle a_{j,2\ell-1}^i, a_{j,2\ell}^i, a_{j+1,\ell}^i \rangle & \text{where } j \in [\log_2(n/2) - 1] \text{ and } \ell \in [n/2^{j+1}] \\ \tilde{b}_{j,\ell}^i: & \langle \tilde{a}_{j,2\ell-1}^i, \tilde{a}_{j,2\ell}^i, \tilde{a}_{j+1,\ell}^i \rangle & \text{where } j \in [\log_2(n/2) - 1] \text{ and } \ell \in [n/2^{j+1}] \\ b_{j,1}^i: & \langle a_{j,1}^i, a_{j,2}^i \rangle & \text{where } j = \log_2(n/2) \\ \tilde{b}_{j,1}^i: & \langle \tilde{a}_{j,1}^i, \tilde{a}_{j,2}^i \rangle & \text{where } j = \log_2(n/2) \end{array}$$

For each edge  $e \in E_{ij}$ ,  $1 \leq i < j \leq k$ , we have the following preferences:

$$\begin{array}{ll} e_1: & \langle e_2, u_{2h+1}, q_{\lceil \ell/2 \rceil}^{ij} \rangle & \text{where for some } \ell \in [m], e = uv = E_{ij}(\ell) \text{ s.t. } u \in V_i \text{ and} \\ & & \text{for some } h \in [r_u], e_1 = \mathcal{E}_u(h) \\ e_2: & \langle e_1, e_3 \rangle & \\ e_3: & \langle e_4, v_{2h+1}, e_2 \rangle & \text{where } e = uv \text{ s.t. } v \in V_j \text{ and} \\ & & \text{for some } h \in [r_v], e_3 = \mathcal{E}_v(h). \\ e_4: & \langle e_3, \tilde{q}_{\lceil \ell/2 \rceil}^{ij} \rangle & \text{where for some } \ell \in [m], e = uv = E_{ij}(\ell) \end{array}$$

For the special vertices associated with  $E_{ij}$ , we have the following preferences:

$$\begin{array}{ll} q_\ell^{ij}: & \langle e_1, e'_1, c_{1,\ell}^{ij} \rangle & \text{where } \ell \in [m/2], e = \sigma(E_{ij}, 2\ell - 1) \text{ and } e' = \sigma(E_{ij}, 2\ell) \\ \tilde{q}_\ell^{ij}: & \langle e_4, e'_4, \tilde{c}_{1,\ell}^{ij} \rangle & \text{where } \ell \in [m/2], e = \sigma(E_{ij}, 2\ell - 1) \text{ and } e' = \sigma(E_{ij}, 2\ell) \\ c_{1,\ell}^{ij}: & \langle q_\ell^{ij}, d_{1,\lceil \ell/2 \rceil}^{ij} \rangle & \text{where } \ell \in [m/2] \\ \tilde{c}_{1,\ell}^{ij}: & \langle \tilde{q}_\ell^{ij}, \tilde{d}_{1,\lceil \ell/2 \rceil}^{ij} \rangle & \text{where } \ell \in [m/2] \\ c_{h,\ell}^{ij}: & \langle d_{h-1,\ell}^{ij}, d_{h,\lceil \ell/2 \rceil}^{ij} \rangle & \text{where } h \in [\log_2(m/2)] \setminus \{1\}, \ell \in [m/2^h] \\ \tilde{c}_{h,\ell}^{ij}: & \langle \tilde{d}_{h-1,\ell}^{ij}, \tilde{d}_{h,\lceil \ell/2 \rceil}^{ij} \rangle & \text{where } h \in [\log_2(m/2)] \setminus \{1\} \text{ and } \ell \in [m/2^h] \\ d_{h,\ell}^{ij}: & \langle c_{h,2\ell-1}^{ij}, c_{h,2\ell}^{ij}, c_{h+1,\ell}^{ij} \rangle & \text{where } h \in [\log_2(m/2) - 1] \text{ and } \ell \in [m/2^{h+1}] \\ \tilde{d}_{h,\ell}^{ij}: & \langle \tilde{c}_{h,2\ell-1}^{ij}, \tilde{c}_{h,2\ell}^{ij}, \tilde{c}_{h+1,\ell}^{ij} \rangle & \text{where } h \in [\log_2(m/2) - 1] \text{ and } \ell \in [m/2^{h+1}] \\ d_{h,1}^{ij}: & \langle c_{h,1}^{ij}, c_{h,2}^{ij} \rangle & \text{where } h = \log_2(m/2) \\ \tilde{d}_{h,1}^{ij}: & \langle \tilde{c}_{h,1}^{ij}, \tilde{c}_{h,2}^{ij} \rangle & \text{where } h = \log_2(m/2) \end{array}$$

## 24:10 On the (Parameterized) Complexity of Almost Stable Marriage

Throughout this section we assume that there exists solution  $\eta$ , and everything will be defined with respect to  $\eta$ . In fact, we assume that  $\eta$  is a *hypothetical* solution to  $(G, \mathcal{L}, \mu, k, q, t)$  such that in every component of  $G[V(\mu\Delta\eta)]$ , the number of  $\eta$ -edges is more than the number of  $\mu$ -edges in this component, that is,  $\eta$  satisfies the property specified in Lemma 5.

We start by defining a notion of good coloring.

► **Definition 6.** A function  $f : V(G) \rightarrow \{0, 1\}$  is called a good coloring, if the following properties are satisfied.

1. Every vertex in  $V(\mu\Delta\eta)$  is colored 1.
2. Let **border** be the set of neighbors of the vertices in  $V(\mu\Delta\eta)$  outside the set  $V(\mu\Delta\eta)$ , that is,  $\text{border} = N_G(V(\mu\Delta\eta))$ , and **bordermates** be the set of matching partners (if they exist) of the vertices in **border** in  $\mu$ . Every vertex in  $\text{border} \cup \text{bordermates}$  is colored 2.

We will show that a random function  $f$  that assigns each vertex of the graph  $G$  independently with color 1 or 2 with probability<sup>5</sup>  $1/2$  each is a good function with probability depending only on  $q$  and  $d$ . In particular, we can say the following about  $f$ .

- Every vertex in  $V(\mu\Delta\eta)$  is colored 1 w.p. at least  $\frac{1}{2^{2q}}$ .
- Every vertex in  $\text{border} \cup \text{bordermates}$  is colored 2 w.p. at least  $\frac{1}{2^{4qd}}$ . To see this, note that  $|V(\mu\Delta\eta)| \leq q$  and the maximum degree of a vertex in the graph  $G$  is  $d$ , and so  $|\text{border} \cup \text{bordermates}| \leq 2|\text{border}| = 2|N_G(V(\mu\Delta\eta))| \leq 4qd$ .

For each  $i \in [2]$ , let  $V_i$  denotes the set of vertices of the graph  $G$  that are colored  $i$  using the function  $f$ . Summarizing the above mentioned properties we get the following.

► **Lemma 7.** Let  $V_1, V_2$ , **border** and **bordermates** be as defined above. Then, w.p. at least  $\frac{1}{2^{2q+4qd}}$ ,  $V(\mu\Delta\eta) \subseteq V_1$  and  $\text{border} \cup \text{bordermates} \subseteq V_2$ . Thus,  $f$  is a good coloring w.p. at least  $\frac{1}{2^{2q+4qd}}$ .

Due to Lemma 7, we have the following:

► **Corollary 8.** Every component in  $G[V(\mu\Delta\eta)]$  is a component in  $G[V_1]$  w.p. at least  $\frac{1}{2^{2q+4qd}}$ .

The proof of Corollary 8 follows from the fact that  $V(\mu\Delta\eta) \subseteq V_1$  and  $\text{border} = N_G(V(\mu\Delta\eta))$  is a subset of  $V_2$  w.p. at least  $\frac{1}{2^{2q+4qd}}$ . Due to Corollary 8, if there exists a component  $C$  in  $G[V_1]$  containing a vertex  $u \in V(G)$  that is saturated in  $\mu$ , such that  $\mu(u) \notin C$ , then  $C$  is not a component in  $G[V(\mu\Delta\eta)]$ . This leads to the following definition. A component  $C$  in  $G[V_1]$  is called a *colored-component*, if for every vertex  $v \in C$ , we have that  $\mu(v) \in C$ . Thus, we get the following lemma.

► **Lemma 9.** Let  $G$  be a graph and  $f : V(G) \rightarrow \{0, 1\}$  be a good function. Then, every component  $C$  of  $G[V(\mu\Delta\eta)]$  is also a component of  $G[V_1]$  and further it is a colored-component.

Let  $(G, f)$  be a pair such that  $G$  is the input graph and  $f$  is a good coloring function on  $V(G)$ . We call such  $(G, f)$  as a *colored instance*.

In light of Corollary 8, to find  $\mu\Delta\eta$ , in Phase II, we color the edges of  $G[V_1]$  in order to identify the components of the graph that *only contain edges of  $\mu\Delta\eta$* . Let  $G_1 = G[V_1]$  and  $G' = G_1[V(\mu\Delta\eta)]$ .

<sup>5</sup> Henceforth, we will use the shortened form w.p. for “with probability”.

**Phase II: Edge Separation.** We first define a notion of *edge-colored instance*.

► **Definition 10.** Let  $f : V(G) \rightarrow \{0, 1\}$  and  $g : E(G) \rightarrow \{\text{Red}, \text{Green}, \text{Blue}\}$  be two functions. An instance  $(G, f, g)$  is called an *edge-colored instance* if the following properties are satisfied.

1.  $(G, f)$  is a colored instance.
2. Every edge in  $\mu\Delta\eta$  is colored **Red**.
3. Every edge in  $E(G') \setminus (\mu\Delta\eta)$  is colored **Green**.
4. Every edge in  $E(G) \setminus E(G_1)$  is colored **Blue**.

Given a colored instance  $(G, f)$ , we select a function  $g$ , as explained below, such that  $(G, f, g)$  becomes an edge-colored instance with high probability.

Let  $g$  be a function that colors each edge of the subgraph  $G_1$  independently with colors **Red** or **Green** with probability  $1/2$  each. Furthermore,  $g$  colors every edge in  $E(G) \setminus E(G_1)$  with **Blue**.

The following properties hold for the graph  $G_1$  that is colored using the function  $g$ :

- Every edge in  $\mu\Delta\eta$  is colored **Red** with probability at least  $\frac{1}{2^q}$ .
- Every edge in  $E(G') \setminus (\mu\Delta\eta)$  is colored **Green** with probability at least  $\frac{1}{2^{2qd}}$ , because  $|V(\mu\Delta\eta)| \leq 2q$  and  $d$  is the maximum degree of a vertex in the graph  $G$ , so  $|E(G')| \leq 2qd$ .
- Every edge in  $E(G) \setminus E(G_1)$  has been colored **Blue** w.p. 1.

For  $i \in \{\text{Red}, \text{Green}, \text{Blue}\}$ , let  $E_i$  denotes the set of edges of the graph  $G$  that are colored  $i$  using the function  $g$ . Then, due to the above mentioned coloring properties of the graph  $G_1$ , we have the following result.

► **Lemma 11.** Let  $(G, f)$  be a colored instance. Furthermore, let  $G'$ ,  $E_{\text{Red}}$ ,  $E_{\text{Green}}$ , and  $E_{\text{Blue}}$  be as defined above. Then, w.p. at least  $\frac{1}{2^{q+2qd}}$ ,  $\mu\Delta\eta \subseteq E_{\text{Red}}$ ,  $E(G') \setminus (\mu\Delta\eta) \subseteq E_{\text{Green}}$ , and  $E(G) \setminus E(G_1) \subseteq E_{\text{Blue}}$ . Thus,  $(G, f, g)$  is an edge-colored instance w.p. at least  $\frac{1}{2^{q+2qd}}$ .

Note that the edges in  $\mu\Delta\eta$  form vertex-disjoint maximal  $\mu$ -alternating paths/cycles. A component may have *several*  $\mu$ -alternating paths and cycles. Let  $C$  be a colored-component. In what follows, we provide conditions such that if  $C$  satisfies either of them, then they *do not belong* to  $G[V(\mu\Delta\eta)]$ . Such a colored-component is called *malformed*.

1. If the set of **Red** edges in  $C$  do not form vertex disjoint maximal  $\mu$ -alternating paths or cycles, then the component does not belong to  $G[V(\mu\Delta\eta)]$ .
2. Furthermore, due to our assumption on the hypothetical solution  $\eta$ , if the number of **Red** edges in  $C$  that are not in  $\mu$  is at most the number of **Red** edges in  $C$  that are in  $\mu$ , then  $C$  does not belong to  $G[V(\mu\Delta\eta)]$ .
3. If  $C$  does not have any **Red** edge, then it does not belong to  $G[V(\mu\Delta\eta)]$ .

A component  $C$  in  $G_1$  that is not malformed is called an *edge-colored-component* (edge-colored-comp).

The next observation follows from the properties of an edge-colored component.

► **Observation 1.** Let  $(G, f, g)$  be an edge-colored instance. Then, for every edge-colored-comp  $C$  of  $G_1$ , the following holds: (a) The set of **Red** colored edges form a collection of  $\mu$ -alternating path/cycle; and (b) every vertex in  $C$  is incident to at least one **Red** edge.

## 24:12 On the (Parameterized) Complexity of Almost Stable Marriage

Let  $\mathcal{C}_{\text{ecc}}$  be the set of components of  $G_1$  that are edge-colored-comp. In light of Observation 1, our goal is reduced to finding a family of components,  $\mathcal{C}$ , in  $\mathcal{C}_{\text{ecc}}$  that contain the edges of  $\mu \Delta \eta$ . Indeed, to obtain a matching of size at least  $|\mu| + t$ , we need to choose  $t' \leq t$  components of  $G[V_1]$  that have  $\mu$ -augmenting paths (a  $\mu$ -alternating path starting and ending with edges not in  $\mu$ ). However, choosing  $t'$  components arbitrarily might lead to a large number of blocking edges in the solution matching. Thus, to choose the components appropriately, we move to Phase III. In particular we show that if  $(G, f, g)$  is an edge-colored instance, then we can solve the problem in polynomial time.

**Phase III: Size-Fitting with respect to  $g$ .** Let  $(G, \mathcal{L}, \mu, k, q, t)$  be an instance to LS-ASM and  $\eta$  be a hypothetical solution to the problem that satisfies the condition in Lemma 5. Further, let  $(G, f, g)$  be an edge-colored instance and  $\mathcal{C}_{\text{ecc}}$  be the set of components of  $G_1$  that are edge-colored-comp.

We reduce our problem to TWO-DIMENSIONAL KNAPSACK (2D-KP), and after that use an algorithm for 2D-KP, described in Proposition 2, as a subroutine.

TWO-DIMENSIONAL KNAPSACK (2D-KP)

**Input:** A set of tuples,  $\mathcal{X} = \{(a_i, b_i, p_i) \in \mathbb{N}^3 : i \in [n]\}$ , and non-negative integers  $c_1, c_2$  and  $p$ .

**Question:** Does there exist a set  $Z \subseteq [n]$  such that  $\sum_{i \in Z} a_i \leq c_1$ ,  $\sum_{i \in Z} b_i \leq c_2$ , and  $\sum_{i \in Z} p_i \geq p$ ?

► **Proposition 2.** [17] *There exists an algorithm  $\mathcal{A}$  that given an instance  $(\mathcal{X}, c_1, c_2, p)$  of 2D-KP, in time  $\mathcal{O}(nc_1c_2)$ , outputs a solution if it is a YES-instance of 2D-KP; otherwise  $\mathcal{A}$  outputs “no”.*

**Construction 2D-Knapsack.** We construct an instance of 2D-KP as follows. Let  $C_1, \dots, C_\ell$  be the components in  $\mathcal{C}_{\text{ecc}}$ . Intuitively, we construct a family of tuples  $\mathcal{X} = \{(k_i, q_i, t_i) : i \in [\ell]\}$  such that  $k_i$  denotes the number of blocking edges that we encounter if we add edges that are not in  $\mu$  but are present in  $\mu$ -alternating paths/cycles in  $C_i$  to our solution. Similarly,  $q_i$  and  $t_i$  denote the number of edges in the symmetric difference and the increase in the size of the matching due to this alternation operation. By our choice of the components in  $\mathcal{C}_{\text{ecc}}$  all these values are positive integers. Indeed, this is why we selected a hypothetical solution with an additional property. Next, we describe the construction of an instance of 2D-KP.

For each  $i \in [\ell]$ , let  $q_i$  be the number of **Red** colored edges in  $C_i$  and  $t_i = q_i - 2|\mu_i|$  where  $\mu_i$  denotes the edges of  $\mu$  in  $C_i$ . Next, to compute  $k_i$ , for each  $i \in [\ell]$ , we construct a matching  $\xi_i$  as follows. We add all the **Red** colored edges in  $C_i$  that are not in  $\mu$  to  $\xi_i$ . Next, we make another matching  $\Gamma_i$ , that has all the edges in  $\xi_i$ , and additionally, we add all the edges in  $\mu$  to  $\Gamma_i$  whose both endpoints are outside the components in  $\mathcal{C}_{\text{ecc}}$ , and at least one of the endpoints is a neighbor of a vertex in  $C_i$ . Clearly,  $\Gamma_i$  is a matching in the graph  $G$ . To ease notation, we let  $G^i$  denote the graph  $G[V(\Gamma_i) \cup V(C_i) \cup N_G(V(C_i))]$ . We set  $k_i$  as the number of blocking edges with respect to  $\Gamma_i$  in the graph  $G^i$ . Basically, the graph  $G^i$  contains all the vertices in  $C_i$ , their neighbors in **border**, the  $\mu$ -partners of these border vertices in **bordermates**, and the neighbors of  $C_i$  which are unsaturated in  $\mu$ . That is, the number of blocking edges (with respect to  $\Gamma_i$ ) incident on the vertices in the set  $V(\xi_i)$  is  $k_i$  in  $G^i$ . To see this note that there is *no blocking edge* with both endpoints in  $V(\Gamma_i \setminus \xi_i)$

(Proposition 1). The only reason to define  $\Gamma_i$  is to define the value of  $k_i$  in a clean fashion. We next state a simple lemma that shows that no blocking edge is counted twice.

► **Lemma 12** (Locally Pairwise Disjoint Blocking Edges). *Let  $(G, \mathcal{L}, \mu, k, q, t)$  be an instance of LS-ASM and  $(G, f, g)$  be an edge-colored instance. Further, let  $C_i, C_j \in \mathcal{C}_{\text{ecc}}, i \neq j$ , and for  $\ell \in \{i, j\}$ ,  $B_\ell$  denote the set of blocking edges with respect to  $\Gamma_\ell$  in  $G[V(\Gamma_\ell) \cup V(C_\ell) \cup N(V(C_\ell))]$ . Then,  $B_i \cap B_j = \emptyset$ .*

**Proof.** Due to the construction of the matching  $\Gamma_i$ , for all the blocking edges in  $B_i$ , at least one of its endpoints is in  $C_i$ . Similarly, for all the blocking edges in  $B_j$ , at least one of its endpoints is in  $C_j$ . Since  $C_i$  and  $C_j$  are distinct components in  $\mathcal{C}_{\text{ecc}}$ , we infer  $B_i \cap B_j = \emptyset$ . ◀

Let  $\mathcal{X} = \{(k_i, q_i, t_i) : i \in [\ell]\}$ . This completes the construction of an instance  $(\mathcal{X}, k, q, t)$  of 2D-KP. We invoke the algorithm  $\mathcal{A}$  given in Proposition 2 on the instance  $(\mathcal{X}, k, q, t)$  of 2D-KP. If  $\mathcal{A}$  returns a set  $Z$ , then we return “yes”. Otherwise, we report *failure* of the algorithm. It is relatively straightforward to create the solution  $\hat{\eta}$  when the answer is “yes”. Next, we prove the correctness of Phase III.

► **Lemma 13.** *Let  $(G, f, g)$  be an edge-colored instance. Then,  $(\mathcal{X}, k, q, t)$  is a yes-instance of 2D-KP.*

**Proof.** Since  $(G, f, g)$  is an edge-colored instance, due to the definition of edge-colored instance (Definition 10),  $(G, f)$  is a colored-instance. Thus, due to the definition of a colored-instance and Lemma 9, every component in  $G[V(\mu\Delta\eta)]$  is also a component in  $G_1$ .

Clearly, for every component  $C$  in  $G[V(\mu\Delta\eta)]$ , if a vertex  $u \in C$ , then  $\mu(u) \in C$ . Therefore, all the components in  $G[V(\mu\Delta\eta)]$  are colored component. Next, we note that due to Definition 10, all the edges in the set  $\mu\Delta\eta$  are colored **Red**. Thus, for every component  $C$  in  $G[V(\mu\Delta\eta)]$ , **Red** colored edges in  $C$  form vertex disjoint maximal  $\mu$ -alternating paths/cycles. Further, every component  $C$  in  $G[V(\mu\Delta\eta)]$  has at least one **Red** edge. Also, due to our choice of  $\eta$ , the number of **Red** edges in  $C$ , which are not in  $\mu$ , are less than the one that are in  $\mu$ . Therefore, all the components in  $G[V(\mu\Delta\eta)]$  are **edge-colored-comp**. Without loss of generality, let  $C_1, \dots, C_{\hat{\ell}}$  be the components in  $\mathcal{C}_{\text{ecc}}$  that are also in  $G[V(\mu\Delta\eta)]$ . Let us note that  $\mathcal{C}_{\text{ecc}}$  may contain *several other components*. Let  $S = \{i \in [\hat{\ell}] : (k_i, q_i, t_i) \in \mathcal{X}\}$ . We claim that  $S$  is a solution to  $(\mathcal{X}, k, q, t)$ .

Due to the construction of the instance  $(\mathcal{X}, k, q, t)$ , and the facts that  $\eta$  is a solution to  $(G, \mathcal{L}, \mu, k, q, t)$  and  $(G, f, g)$  is an edge-colored instance, clearly,  $\sum_{i \in S} q_i \leq q$  and  $\sum_{i \in S} t_i \geq t$ . We next show that  $\sum_{i \in S} k_i \leq k$ . Recall the definition of  $\xi_i$  and  $\Gamma_i$ . We show that every blocking edge with respect to  $\Gamma_i$  in the graph  $G^i$  is also a blocking edge with respect to  $\eta$  in the graph  $G$ . Let  $uv$  be a blocking edge with respect to  $\Gamma_i$  in the graph  $G^i$ . Then,  $v \succ_u \Gamma_i(u)$  and  $u \succ_v \Gamma_i(v)$ . Due to Proposition 1 and the definition of the matching  $\Gamma_i$ , at least one of the endpoint of the edge  $uv$  is in the component  $C_i$ . Without loss of generality, let  $u \in V(C_i)$ . Since  $C_i$  is also a component in  $G[V(\mu\Delta\eta)]$ , we can infer that  $\eta(u) = \Gamma_i(u)$ . If the vertex  $v$  is also in the component  $C_i$ , then using the same argument as above, we know that  $\eta(v) = \Gamma_i(v)$ . Thus,  $uv$  is also a blocking edge with respect to  $\eta$  in the graph  $G$ . Suppose that  $v \notin V(C_i)$ . Then, since  $v \in V(\Gamma_i) \cup V(C_i)$ ,  $\Gamma_i(v) = \mu(v)$ . Since  $C_i$  is a component in  $G[V(\mu\Delta\eta)]$  and  $u \in V(C_i)$  but  $v \notin C_i$ , we can infer that  $\eta(v) = \mu(v)$ . Since  $u$  and  $v$  have same matching partners in both the matchings  $\eta$  and  $\Gamma_i$ , we can infer that  $uv$  is also a blocking edge with respect to  $\eta$  in the graph  $G$ . Since  $k_i$  is the number of blocking edges with respect to  $\Gamma_i$ , we infer  $\sum_{i \in S} k_i \leq k$ . Hence,  $(\mathcal{X}, k, q, t)$  is a YES-instance of 2D-KP. ◀

► **Lemma 14.** *Suppose that  $(\mathcal{X}, k, q, t)$  is a YES-instance of 2D-KP. Then,  $(G, \mathcal{L}, \mu, k, q, t)$  is a YES-instance of LS-ASM.*

## 24:14 On the (Parameterized) Complexity of Almost Stable Marriage

**Proof.** Suppose that the algorithm  $\mathcal{A}$  in Proposition 2 returns the set  $Z$ . Given the set  $Z$ , we obtain the matching  $\tilde{\eta}$  as follows. Let  $Z(\mathcal{C})$  denotes the family of components in  $\mathcal{C}_{\text{ecc}}$  corresponding to the indices in  $Z$ . Formally,  $Z(\mathcal{C}) = \{C_i \in \mathcal{C}_{\text{ecc}} : i \in Z\}$ . For each component  $C \in Z(\mathcal{C})$ , we add all the **Red** edges in  $C$  that are not in  $\mu$ , to  $\tilde{\eta}$ . That is,  $\tilde{\eta} = \cup_{i \in Z} \xi_i$ . Additionally, we add all the edges in  $\mu$  to  $\tilde{\eta}$  whose both endpoints are outside the components in  $Z(\mathcal{C})$ . We next prove that  $\tilde{\eta}$  is a solution to  $(G, \mathcal{L}, \mu, k, q, t)$ .

▷ **Claim 15.**  $\tilde{\eta}$  is a matching.

**Proof.** Towards the contradiction, suppose that  $uv, uw \in \tilde{\eta}$ , that is, there exists a pair of edges in  $\tilde{\eta}$  that shares an endpoint. Due to Observation 1, in every component of  $Z(\mathcal{C})$ , the **Red** edges form  $\mu$ -alternating path/cycle. Therefore,  $uv$  and  $uw$  both cannot be in a component of  $Z(\mathcal{C})$ . Suppose that  $uv$  is in a component  $C$  in  $Z(\mathcal{C})$ , but  $uw$  does not belong to  $C$ . Then, due to the construction of  $\tilde{\eta}$ ,  $uw \in \mu$ . This contradicts Lemma 9, as  $C$  is a component in  $\mathcal{C}_{\text{ecc}}$ . If  $uv$  and  $uw$  are outside the components in  $Z(\mathcal{C})$ , then due to the construction of  $\tilde{\eta}$ ,  $uv$  and  $uw$  both are in  $\mu$ . This contradicts that  $\mu$  is a matching. ◁

▷ **Claim 16.**  $|\mu \Delta \tilde{\eta}| \leq q$  and  $|\tilde{\eta}| \geq |\mu| + t$ .

**Proof.** Let  $C$  be a component in  $Z(\mathcal{C})$ . Let  $E_{\text{Red}}(C)$  denote the set of **Red** edges in the component  $C$ . For each component  $C_i \in Z(\mathcal{C})$ , let  $\mu_i = \mu \cap E_{\text{Red}}(C_i)$ , that is,  $\mu_i$  is the set of **Red** edges in  $C_i$  that are in  $\mu$ . Let  $\tilde{\mu}$  be the set of edges in  $\mu$  that does not belong to any component in  $Z(\mathcal{C})$ . Thus,  $\mu = \uplus_{C_i \in Z(\mathcal{C})} \mu_i \uplus \tilde{\mu}$ . Due to the construction of  $\tilde{\eta}$ , we have that  $\tilde{\eta} = \uplus_{C_i \in Z(\mathcal{C})} (E_{\text{Red}}(C_i) \setminus \mu_i) \uplus \tilde{\mu}$ . Thus,  $\mu \Delta \tilde{\eta} = \uplus_{C_i \in Z(\mathcal{C})} E_{\text{Red}}(C_i)$ . Hence,

$$|\mu \Delta \tilde{\eta}| = \sum_{C_i \in Z(\mathcal{C})} |E_{\text{Red}}(C_i)| = \sum_{i \in Z} q_i$$

as  $q_i = |E_{\text{Red}}(C_i)|$  for every component  $C_i \in \mathcal{C}_{\text{ecc}}$ . Since  $S$  is a solution to  $(\mathcal{X}, k, q, t)$ ,  $\sum_{i \in Z} q_i \leq q$ . Therefore,  $|\mu \Delta \tilde{\eta}| \leq q$ . Next, we show that  $|\tilde{\eta}| \geq |\mu| + t$ . Due to the construction of  $\tilde{\eta}$ , we know that

$$|\tilde{\eta}| = |\tilde{\mu}| + \sum_{C_i \in Z(\mathcal{C})} |E_{\text{Red}}(C_i) \setminus \mu_i| = |\tilde{\mu}| + \sum_{C_i \in Z(\mathcal{C})} (q_i - |\mu_i|) = |\tilde{\mu}| + \sum_{C_i \in Z(\mathcal{C})} (t_i + |\mu_i|)$$

as  $t_i = q_i - 2|\mu_i|$ . Since  $\sum_{i \in Z} t_i \geq t$ , we obtained that  $|\tilde{\eta}| \geq |\mu| + t$ . ◁

▷ **Claim 17.** There are at most  $k$  blocking edges with respect to  $\tilde{\eta}$ .

**Proof.** Due to the construction of the matching  $\tilde{\eta}$  and Proposition 1, we know that if  $uv$  is a blocking edge with respect to  $\tilde{\eta}$ , then at least one of its endpoint, that is vertex  $u$  or  $v$ , is in  $C_i$ , for some  $C_i \in Z(\mathcal{C})$ . Without loss of generality, let the vertex  $u$  is in the component  $C_i$ . Then, due to the definition of the matching  $\Gamma_i$  and the construction of the matching  $\tilde{\eta}$ , we have that  $\tilde{\eta}(u) = \Gamma_i(u)$ . Now, if  $v$  is also in the component  $C_i$ , then using the same argument  $\tilde{\eta}(v) = \Gamma_i(v)$ . Suppose that  $v$  is not in the component  $C_i$ , then its  $\mu$ -partner, that is  $\mu(v)$  is also not present in  $\mathcal{C}_{\text{ecc}}$  due the the definition of colored-components. Thus,  $\tilde{\eta}(v) = \Gamma_i(v) = \mu(v)$ . Since the matching partners of  $u$  and  $v$  are same in both the matchings  $\eta$  and  $\Gamma_i$ , we have  $uv$  is also a blocking edge with respect to matching  $\Gamma_i$ . Thus, every blocking edge with respect to  $\tilde{\eta}$  is also a blocking edge with respect to  $\Gamma_i$ , for some  $C_i \in Z(\mathcal{C})$ . Recall that  $k_i$  is the number of blocking edges with respect to  $\Gamma_i$  in  $G^i$ . Therefore, the number of blocking edges with respect to  $\tilde{\eta}$  is at most  $\sum_{i \in Z} k_i \leq k$ . ◁

Due to Claims 15, 16, and 17, we can infer that  $\tilde{\eta}$  is a solution to  $(G, \mathcal{L}, \mu, k, q, t)$ . ◀

Due to Lemmas 7 and 11, we obtain a polynomial-time randomized algorithm for LS-ASM which succeeds with probability  $\frac{1}{2^{3q+6qd}}$ . Therefore, by repeating the algorithm independently  $2^{3q+6dq}$  times, where  $n$  is the number of vertices in the graph, we obtain the following result:

► **Theorem 18.** *There exists a randomized algorithm that given an instance of LS-ASM runs in  $2^{3q+6dq}n^{\mathcal{O}(1)}$  time, where  $n$  is the number of vertices in the given graph, and either reports a failure or outputs “yes”. Moreover, if the algorithm is given a YES-instance of the problem, then it returns “yes” with a constant probability.*

**Proof.** Let  $(G, \mathcal{L}, \mu, k, q, t)$  be an instance to LS-ASM and  $\eta$  be a hypothetical solution to the problem. If  $(G, \mathcal{L}, \mu, k, q, t)$  is a YES-instance then by Lemmas 7 and 11, we can get an edge-colored instance,  $(G, f, g)$ , w.p. at least  $\frac{1}{2^{3q+6qd}}$ . Given an edge-colored instance  $(G, f, g)$ , we apply **Construction 2D-Knapsack** and construct an instance of 2D-KP with a family of tuples  $\mathcal{X} = \{(k_i, q_i, t_i) : i \in [\ell]\}$ . Here,  $(\mathcal{X}, k, q, t)$  is a yes-instance of 2D-KP. We can solve the instance in polynomial time using Proposition 2. Correctness of this step follows from Lemmas 13 and 14. Thus, if  $(G, \mathcal{L}, \mu, k, q, t)$  is a YES-instance, then we return that it is a YES-instance with probability at least  $\frac{1}{2^{3q+6qd}}$ . Indeed, if  $(G, \mathcal{L}, \mu, k, q, t)$  is a NO-instance, then we return that it is a NO-instance with probability 1. Thus, to boost the success probability to a constant, we repeat the algorithm independently  $2^{3q+6dq}(\log n)^{\mathcal{O}(1)}$  times, where  $n$  is the number of vertices in  $G$ . Indeed, the success probability is at least

$$1 - \left(1 - \frac{1}{2^{3q+6qd}}\right)^{2^{3q+6dq}(\log n)^{\mathcal{O}(1)}} \geq 1 - \frac{1}{n^{\mathcal{O}(1)}}.$$

This concludes the proof. ◀

The derandomization of the algorithm is in the full version.

## 5 In Conclusion

In this paper, we initiated the study of the computational complexity of the trade-off between size and stability through the lenses of both multivariate analysis and local search. Since ASM is NP-hard for a graph in which every vertex has degree at most three, the natural question that arises here is: Is ASM polynomial-time solvable for the graph in which every vertex has degree at most two? It is worth mentioning that there is a fairly straightforward dynamic programming algorithm that solves this question in polynomial time. The basis idea is as follows. This graph, quite clearly, is a disjoint union of paths and cycles.

Consider a hypothetical solution  $\eta$  in the path or cycle  $X_n$  in  $G$ . Suppose that we know the submatching of  $\eta$ , call it  $\eta'$ , that is contained in a subpath of  $X_n$  as well as the subset of blocking edges with respect to  $\eta$  that are in this subpath. Then, we can extend  $\eta'$  to  $\eta$  by keeping all the necessary partial solutions. We can briefly sketch this idea as follows. Suppose that  $\eta_i$  is a matching in the subpath  $P_i = (1, \dots, i)$ . We want to extend  $\eta_i$  for the subpath  $P_{i+1}$ . If  $v_i$  is saturated in  $\eta_i$ , then we cannot add edge  $v_i v_{i+1}$  to  $\eta_{i+1}$  and we can easily check if it is a blocking edge with respect to  $\eta_{i+1}$  in  $P_{i+1}$ . If  $v_i$  is unsaturated in  $\eta_i$ , then we have two possibilities: edge  $v_i v_{i+1}$  is and is not in  $\eta_{i+1}$ . If it is, then we can check if  $v_i v_{i-1}$  is a blocking edge with respect to  $\eta_{i+1}$  in  $P_{i+1}$ . Otherwise,  $v_i v_{i+1}$  is a blocking edge with respect to  $\eta_{i+1}$  in  $P_{i+1}$ . All these possibilities can be taken care by appropriately defining the table entries. In the table, however, we do not need to store the whole matching. We only need to remember the matching partner of vertices, such as  $v_{i-1}$ , as that will help in checking if  $v_i v_{i+1}$  is a blocking edge. We can similarly argue for a cycle in  $G$ .

Next, we would like to point out that our hardness results, that is, Theorems 1, 2, and 3 hold even when the preference lists of the vertices in each side of the partition respect a master ordering of vertices i.e., the relative ordering of the vertices in a preference list is same as that of a fixed ordering of all the vertices on the other side. We discuss it in details in the full version of the paper.

**Future work.** We conclude the paper with a few directions for further research.

- In certain scenarios, the “satisfaction” of the agents (there exist several measures such as *egalitarian*, *sex-equal*, *balance*) might be of importance. Then, it might be of interest to study the tradeoff between  $t$  and  $k$ , tradeoff between egalitarian/sex-equal/balance cost and  $k$ .
- The formulation of ASM can be generalized to the case where the input contains a utility function on the edges and the objective is to maximize the value of a solution matching subject to this function.

---

## References

- 1 D. J. Abraham, P. Biró, and D. F. Manlove. “Almost stable” matchings in the roommates problem. In *International Workshop on Approximation and Online Algorithms (WG)*, pages 1–14, 2005.
- 2 P. Biró, T. Fleiner, R. W. Irving, and D. F. Manlove. The college admissions problem with lower and common quotas. *Theoretical Computer Science*, 411(34-36):3136–3153, 2010.
- 3 P. Biró, D. Manlove, and S. Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411(16-18):1828–1841, 2010.
- 4 P. Biró, D. F. Manlove, and E. J. McDermid. “Almost stable” matchings in the roommates problem with bounded preference lists. *Theoretical Computer Science*, 432:10–20, 2012.
- 5 J. Chen, D. Hermelin, M. Sorge, and H. Yedidsion. How hard is it to satisfy (almost) all roommates? In *International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 35:1–35:15, 2018.
- 6 J. Chen, P. Skowron, and M. Sorge. Matchings under preferences: Strength of stability and trade-offs. In *ACM Conference on Economics and Computation (EC)*, pages 41–59, 2019.
- 7 Á. Cseh and D. F. Manlove. Stable marriage and roommates problems with restricted edges: Complexity and approximability. *Discrete Optimization*, 20:62–89, 2016.
- 8 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 10 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 11 M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. A. Rosamond, S. Saurabh, and Y. Villange. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012.
- 12 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.
- 13 F. V. Fomin, D. Lokshtanov, F. Panolan, and S. Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.
- 14 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

- 15 D. Gusfield and R. W. Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.
- 16 R. W. Irving and D. Manlove. Finding large stable matchings. *ACM Journal of Experimental Algorithmics*, 2009.
- 17 H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.
- 18 S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM journal on computing*, 32(2):470–487, 2003.
- 19 D. E. Knuth. Marriages stables. *Technical report*, 1976.
- 20 A. Krokhn and D. Marx. On the hardness of losing weight. *ACM Transactions on Algorithms*, 8(2):1–18, 2012.
- 21 D. Manlove. *Algorithmics of matching under preferences*, volume 2. World Scientific, 2013.
- 22 D. F. Manlove, I. McBride, and J. Trimble. "Almost-stable" matchings in the Hospitals/Residents problem with couples. *Constraints*, 22(1):50–72, 2017.
- 23 D. Marx. Local search. *Parameterized Complexity News*, 3:7–8, 2008.
- 24 D. Marx. Searching the k-change neighborhood for TSP is  $W[1]$ -hard. *Operations Research Letters*, 36(1):31–36, 2008.
- 25 D. Marx and I. Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010.
- 26 D. Marx and I. Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011.
- 27 S. Micali and V. V. Vazirani. An  $O(\sqrt{|V||E|})$  algorithm for finding maximum matching in general graphs. In *Foundations of Computer Science (FOCS)*, pages 17–27, 1980.
- 28 R. Neidermeier. *Invitation to fixed-parameter algorithms*. Springer, 2006.
- 29 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003.
- 30 A. E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6), 1984.
- 31 A. E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica: Journal of the Econometric Society*, 54(2):425–427, 1986.
- 32 S. Szeider. The parameterized complexity of k-flip local search for sat and max sat. *Discrete Optimization*, 8(1):139–145, 2011.
- 33 El-Ghazali Talbi. *Metaheuristics: From design to implementation*, volume 74. John Wiley & Sons, 2009.
- 34 K. Tomoeda. Finding a stable matching under type-specific minimum quotas. *Journal of Economic Theory*, 176:81–117, 2018.



# Min-Cost Popular Matchings

Telikepalli Kavitha

Tata Institute of Fundamental Research, Mumbai, India  
kavitha@tifr.res.in

---

## Abstract

Let  $G = (A \cup B, E)$  be a bipartite graph on  $n$  vertices where every vertex ranks its neighbors in a strict order of preference. A matching  $M$  in  $G$  is *popular* if there is *no* matching  $N$  such that vertices that prefer  $N$  to  $M$  outnumber those that prefer  $M$  to  $N$ . Popular matchings always exist in  $G$  since every stable matching is popular. Thus it is easy to find a popular matching in  $G$  – however it is NP-hard to compute a *min-cost* popular matching in  $G$  when there is a *cost* function on the edge set; moreover it is NP-hard to approximate this to any multiplicative factor. An  $O^*(2^n)$  algorithm to compute a min-cost popular matching in  $G$  follows from known results. Here we show:

- an algorithm with running time  $O^*(2^{n/4}) \approx O^*(1.19^n)$  to compute a min-cost popular matching;
- assume all edge costs are non-negative – then given  $\varepsilon > 0$ , a randomized algorithm with running time  $\text{poly}(n, \frac{1}{\varepsilon})$  to compute a matching  $M$  such that  $\text{cost}(M)$  is at most twice the optimal cost and with high probability, the fraction of all matchings more popular than  $M$  is at most  $\frac{1}{2} + \varepsilon$ .

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Bipartite graphs, Stable matchings, Dual certificates

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.25

**Funding** We acknowledge support of the Department of Atomic Energy, Government of India, under project no. RTI4001.

**Acknowledgements** Thanks to Jannik Matuschke for conversations on semi-popular matchings and to Piyush Srivastava for helpful discussions on sampling matchings. Thanks to the reviewers for their helpful comments.

## 1 Introduction

Consider a matching problem in a bipartite graph  $G = (A \cup B, E)$  on  $n$  vertices where every vertex has a strict ranking of its neighbors. Matching  $M$  is stable if  $M$  admits no *blocking edge* – an edge  $(a, b)$  is a blocking edge to  $M$  if  $a$  and  $b$  prefer each other to their respective assignments in  $M$ . Stable matchings always exist in  $G$  and one such matching can be computed in linear time by the classical Gale-Shapley algorithm [14]. Suppose there is a cost function on the edge set  $E$ . Computing a min-cost stable matching in  $G$  is a well-studied problem and there are several polynomial time algorithms to compute a min-cost stable matching and special variants of this problem [10, 11, 12, 21, 29, 30, 31].

Stability or absence of blocking edges is a rather strict notion – it is known that all stable matchings have the same size and match the same subset of vertices [15]. Consider the instance  $G = (A \cup B, E)$  where  $A = \{a_1, a_2\}$ ,  $B = \{b_1, b_2\}$ , and  $E = \{(a_1, b_1), (a_1, b_2), (a_2, b_1)\}$ . Suppose  $a_1$  prefers  $b_1$  to  $b_2$  and similarly,  $b_1$  prefers  $a_1$  to  $a_2$ . The only stable matching here is  $\{(a_1, b_1)\}$  whose size is half the size of the perfect matching  $\{(a_1, b_2), (a_2, b_1)\}$ .

**A relaxation.** In applications such as matching students to advisers, we would like to replace the notion of “no blocking edges” with a more relaxed notion of stability for the sake of obtaining a larger matching, or more generally, a more optimal matching. A natural relaxation of stability is the notion of *popularity* introduced by Gärdenfors [16] in 1975.



© Telikepalli Kavitha;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 25; pp. 25:1–25:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 25:2 Min-Cost Popular Matchings

Roughly speaking, a matching is popular if there is no matching that makes more vertices happier. More formally, we say a vertex  $u \in A \cup B$  prefers matching  $M$  to matching  $N$  if either (i)  $u$  is matched in  $M$  and unmatched in  $N$  or (ii)  $u$  is matched in both  $M$  and  $N$  and  $u$  prefers its partner in  $M$  to its partner in  $N$ . For any two matchings  $M$  and  $N$ , let  $\phi(M, N)$  be the number of vertices that prefer  $M$  to  $N$ .

► **Definition 1.** A matching  $M$  is popular if  $\phi(M, N) \geq \phi(N, M)$  for every matching  $N$  in  $G$ , i.e.,  $\Delta(M, N) \geq 0$  where  $\Delta(M, N) = \phi(M, N) - \phi(N, M)$ .

In an election between  $M$  and  $N$  where vertices cast votes,  $\phi(M, N)$  is the number of votes for  $M$  and  $\phi(N, M)$  is the number of votes for  $N$ . A popular matching never loses an election against another matching: thus it is a weak *Condorcet winner* [3, 4] in the corresponding voting instance. Although (weak) Condorcet winners need not exist in a general voting instance, popular matchings always exist in a bipartite graph since every stable matching is popular [16]. In fact, a stable matching is a min-size popular matching [19]. In the example described earlier, the perfect matching  $\{(a_1, b_2), (a_2, b_1)\}$  is unstable but popular.

Efficient algorithms are known to compute a max-size popular matching in  $G$  [19, 23]. Though computing a min-size/max-size popular matching is easy, surprisingly, it is NP-hard to decide if  $G$  admits a popular matching that is not a min-size/max-size popular matching [9]. Also, computing a min-cost popular matching is NP-hard [9] when there is a cost function on the edge set. The min-cost popular matching problem includes other optimization problems such as computing a popular matching with forced/forbidden edges or one with max-utility as special cases and these variants are also NP-hard [9].

In applications such as matching students to advisers or medical residents to hospitals, where matchings have a long-term impact, it may be worthwhile to spend (exponential) time and compute an optimal popular matching in  $G$ . It follows from recent work on finding popular matchings in non-bipartite graphs [25] that there is an  $O^*(2^n)$  time algorithm for the min-cost popular matching problem in a bipartite graph on  $n$  vertices (note that  $O^*(2^n)$  stands for  $O(2^n \cdot \text{poly}(n))$ ). Here we study faster exponential time algorithms for this problem and show the following result.

► **Theorem 2.** Given a bipartite graph  $G = (A \cup B, E)$  on  $n$  vertices where every vertex has a strict preference list ranking its neighbors and a function  $\text{cost} : E \rightarrow \mathbb{R}$ , a min-cost popular matching in  $G$  can be computed in  $O^*(2^{n/4}) \approx O^*(1.19^n)$  time.

The running time of our algorithm is  $O(2^p \cdot \text{poly}(n))$  where  $p$  is the number of connected components of size at least 4 in a special subgraph of  $G$ . Thus our algorithm is an FPT algorithm parameterized by  $p$  and when  $p = O(\log n)$ , this is a polynomial time algorithm.

When edge costs are non-negative, the *max-cost* popular matching problem in  $G$  admits an efficient  $1/2$ -approximation algorithm – however the *min-cost* popular matching problem is NP-hard to approximate within any multiplicative factor even when edge costs are in  $\{0, 1\}$  [9]. This motivates the following question: when edge costs are non-negative, is there an efficient algorithm to compute an *approximately popular* matching whose cost is  $O(\text{opt})$ , where  $\text{opt}$  is the cost of a min-cost popular matching?

There are several ways to define an approximately popular matching and we choose the following novel definition: a matching  $M$  such that  $\phi(M, N) \geq \phi(N, M)$  for a majority of matchings  $N$  in  $G$ . This motivates the definition of a *semi-popular* matching as follows.

► **Definition 3.** Call a matching  $M$  in  $G = (A \cup B, E)$  semi-popular if  $\phi(M, N) \geq \phi(N, M)$  for at least half the matchings  $N$  in  $G$ .

Though semi-popularity is a natural relaxation of popularity, the set of semi-popular matchings seems to lack the structure of the set of popular matchings. We do not know how to efficiently test if a given matching is semi-popular or not. We show the following result on computing an *almost semi-popular* matching in an instance  $G$  with non-negative edge costs.

► **Theorem 4.** *Given a bipartite graph  $G = (A \cup B, E)$  with  $\text{cost} : E \rightarrow \mathbb{R}_{\geq 0}$  and  $\varepsilon > 0$ , a matching  $M$  can be computed in  $\text{poly}(n, \frac{1}{\varepsilon})$  time such that  $\text{cost}(M) \leq 2\text{opt}$  and with high probability  $M$  is undefeated by at least  $1/2 - \varepsilon$  fraction of all matchings in  $G$ .*

Using the notation of bi-criteria approximation algorithms (see [27]), the above result is with high probability a  $(2, \frac{1}{2} - \varepsilon)$  approximation of a min-cost popular matching, where the first coordinate is the ratio of the cost of our matching and  $\text{opt}$  and the second coordinate is a measure of popularity of our matching, more precisely, it is the fraction of matchings in  $G$  that our matching does not lose to. Designing an efficient algorithm to compute an  $(O(1), 1 - \varepsilon)$  bi-criteria approximation is an open problem.

## 1.1 Background and Related Results

Algorithmic questions in the domain of popular matchings have been studied in the last 10-15 years. We refer to [5] for a survey. Initially, algorithms for popular matchings in instances with *one-sided* preferences (only vertices in  $A$  have preferences) were studied [1]. In the domain of two-sided preferences with ties, it is NP-complete to decide if popular matchings exist or not [2, 6]. The problem of deciding if a non-bipartite graph with strict preferences admits a popular matching is NP-complete [9, 17]. Popular matchings always exist in bipartite graphs with strict preferences [16]. However, as mentioned earlier, it is NP-hard to compute or approximate a min-cost popular matching. In order to cope with this hardness of approximation, a relaxation of popularity called *quasi-popularity* was considered in [8].

A matching  $M$  is quasi-popular if  $\phi(N, M) \leq 2 \cdot \phi(M, N)$  for all matchings  $N$ . That is,  $M$  may lose many elections, however the *factor* of defeat, i.e., the ratio of number of votes won by the rival matching and the number of votes won by  $M$ , is bounded by 2. On the other hand, a semi-popular matching does not lose too many elections. A polynomial time algorithm to compute a quasi-popular matching of cost at most  $\text{opt}$  was given in [8].

There is a vast literature on fast exponential time algorithms for NP-hard problems and we refer to the book [13] on this subject. An algorithm with running time  $O^*(c^n)$ , where  $c = O(1)$ , was given in [25] to decide if a non-bipartite graph on  $n$  vertices with strict preferences has a popular matching or not. Fast exponential time algorithms for other hard problems in matchings under preferences are also known, e.g., the sex-equal stable marriage problem in bipartite graphs where the objective is to find a *fair* stable matching – a fast exponential time algorithm is known for this problem when the length of preference lists of vertices on one side of the bipartite graph is bounded from above by a small value [28].

## 1.2 Techniques

An  $O^*(2^n)$  time algorithm was given in [25] to decide if a special popular matching called a *truly popular* matching exists or not in a general graph (not necessarily bipartite) on  $n$  vertices. A truly popular matching is a matching that is *popular fractional* (defined in Section 4). In bipartite graphs, every popular matching is truly popular and so this algorithm leads to an algorithm with running time  $O^*(2^n)$  to compute a min-cost popular matching in the bipartite graph  $G$ . Our faster exponential time algorithm is an extension of this algorithm.

**The earlier algorithm.** The  $O^*(2^n)$  time algorithm uses dual certificates or *witnesses* for popular matchings, where a witness  $\vec{\alpha}$  is a vector in  $\{0, \pm 1\}^n$  that obeys certain constraints (see Theorem 5). Corresponding to each of the  $2^n$  parity combinations – whether  $\alpha_u$  is 0 or  $\pm 1$  for each vertex  $u$  – the  $O^*(2^n)$  algorithm constructs a stable matching instance and shows that every stable matching in this instance that avoids certain edges maps to a popular matching in  $G$ . Conversely, every popular matching in  $G$  can be realized as a stable matching that avoids certain edges in one of these  $2^n$  instances. Computing a min-cost stable matching that excludes certain edges in each of these  $2^n$  instances and taking the least cost such matching leads us to a min-cost popular matching in  $G$ .

**Our faster algorithm.** It was shown in [8] that all vertices in the same connected component in a subgraph  $G_0$  of  $G$  called its “popular subgraph” have the same parity of their  $\alpha$ -values. So instead of considering individual vertices, we consider non-trivial connected components in  $G_0$  as our “units”. Our main idea is that it suffices for the algorithm to go through parity combinations of  $\alpha$ -values only for connected components in  $G_0$  of size at least 4. So our algorithm constructs at most  $2^{n/4}$  stable matching instances. However our stable matching instances are more elaborate than in the earlier algorithm and the most technical part of the analysis is the proof that stable matchings that avoid certain edges in such an instance map to popular matchings in  $G$ . The algorithm and its proof of correctness are given in Section 3.

**Our bi-criteria approximation algorithm.** Unlike the popular matching polytope, the popular fractional matching polytope has a compact extended formulation [26]. Thus a min-cost popular fractional matching can be computed in polynomial time by linear programming over this polytope. It is known that this polytope is half-integral [20]. Thus we can efficiently find *two* matchings  $M_1, M_2$  in  $G$  such that  $(I_{M_1} + I_{M_2})/2$  is a min-cost popular fractional matching in  $G$ , where  $I_M$  is the edge incidence vector of matching  $M$ . This implies that one of  $M_1, M_2$  is semi-popular.

Interestingly, we do not know how to efficiently decide which of  $M_1, M_2$  is semi-popular. We use the random sampler from [22] to sample matchings from a distribution close to the uniform distribution – this allows us to decide with high probability whether both  $M_1$  and  $M_2$  are almost semi-popular or one of them is not. This result is given in Section 4.

## 2 Popular Matchings and Witnesses

Let  $\tilde{G}$  be the graph  $G$  augmented with self-loops. We assume that each vertex is its own last choice neighbor. Any matching  $M$  in  $G$  can henceforth be regarded as a perfect matching  $\tilde{M}$  in  $\tilde{G}$  by adding self-loops for all vertices left unmatched in  $M$ . The following edge weight function  $\text{wt}_M$  in  $\tilde{G}$  will be useful to us. For any edge  $(a, b)$  in  $G$ , define:

$$\text{wt}_M(a, b) = \begin{cases} 2 & \text{if } (a, b) \text{ is a blocking edge to } M; \\ -2 & \text{if both } a \text{ and } b \text{ prefer their respective partners in } M \text{ to each other;} \\ 0 & \text{otherwise.} \end{cases}$$

So  $\text{wt}_M(e) = 0$  for every edge  $e \in M$ . We need to define  $\text{wt}_M$  on self-loops also. For any vertex  $u \in A \cup B$ , let  $\text{wt}_M(u, u) = 0$  if  $(u, u) \in \tilde{M}$ , else  $\text{wt}_M(u, u) = -1$ . Let  $\tilde{E} = E \cup \{(u, u) : u \in A \cup B\}$ . For any matching  $N$  in  $G$ , we have:

$$\text{wt}_M(\tilde{N}) = \sum_{e \in \tilde{N}} \text{wt}_M(e) = \phi(N, M) - \phi(M, N) = \Delta(N, M).$$

Hence  $M$  is popular in  $G$  if and only if every perfect matching in the graph  $\tilde{G}$  (with edge weights given by  $\text{wt}_M$ ) has weight at most 0. Consider the max-weight perfect matching LP in the graph  $\tilde{G}$ : this is (LP1) given below in variables  $x_e$  for  $e \in \tilde{E}$ . Here  $\tilde{\delta}(u) = \delta(u) \cup \{(u, u)\}$  for  $u \in A \cup B$ . The linear program (LP2) in variables  $\alpha_u$  for  $u \in A \cup B$  is the dual LP.

$$\begin{array}{ll} \max \sum_{e \in \tilde{E}} \text{wt}_M(e) \cdot x_e & \text{(LP1)} \\ \text{s.t.} & \sum_{e \in \tilde{\delta}(u)} x_e = 1 \quad \forall u \in A \cup B \\ & x_e \geq 0 \quad \forall e \in \tilde{E}. \end{array} \quad \begin{array}{ll} \min \sum_{u \in V} \alpha_u & \text{(LP2)} \\ \text{s.t.} & \alpha_a + \alpha_b \geq \text{wt}_M(a, b) \quad \forall (a, b) \in E \\ & \alpha_u \geq \text{wt}_M(u, u) \quad \forall u \in A \cup B \end{array}$$

The characterization of popular matchings given in Theorem 5 follows from LP-duality and total unimodularity of the system. Recall that  $|A \cup B| = n$ .

► **Theorem 5** ([24, 26]). *A matching  $M$  in  $G = (A \cup B, E)$  is popular if and only if there exists a vector  $\vec{\alpha} \in \{0, \pm 1\}^n$  such that  $\sum_{u \in A \cup B} \alpha_u = 0$ ,*

$$\alpha_a + \alpha_b \geq \text{wt}_M(a, b) \quad \forall (a, b) \in E \quad \text{and} \quad \alpha_u \geq \text{wt}_M(u, u) \quad \forall u \in A \cup B.$$

**Proof.** The linear program (LP2) admits an optimal solution that is integral since its constraint matrix is totally unimodular. The vector  $\vec{\alpha}$  is an integral optimal solution of (LP2). We have  $\alpha_u \geq \text{wt}_M(u, u) \geq -1$  for all  $u$ .

Since  $\vec{M}$  is an optimal solution to (LP1), complementary slackness implies  $\alpha_u + \alpha_v = \text{wt}_M(u, v) = 0$  for each edge  $(u, v) \in M$ . Thus  $\alpha_u = -\alpha_v \leq 1$  for every vertex  $u$  matched to a non-trivial neighbor  $v$  in  $M$ . Regarding any vertex  $u$  such that  $(u, u) \in \vec{M}$ , we have  $\alpha_u = \text{wt}_M(u, u) = 0$  (by complementary slackness). Hence  $\vec{\alpha} \in \{0, \pm 1\}^n$ . ◀

► **Definition 6.** *For any popular matching  $M$ , a vector  $\vec{\alpha} \in \{0, \pm 1\}^n$  as given in Theorem 5 is called a witness of  $M$ .*

A popular matching may have several witnesses. A stable matching  $S$  has  $\vec{0}$  as a witness, since  $\text{wt}_S(e) \leq 0$  for all edges  $e$  in  $\tilde{G}$ . Call an edge  $e$  in  $G = (A \cup B, E)$  *popular* if there is some popular matching in  $G$  that contains  $e$ . Let  $E_0$  be the set of popular edges in  $G$ . The set  $E_0$  can be computed in linear time [7]. Call the subgraph  $G_0 = (A \cup B, E_0)$  the *popular subgraph* of  $G$ . The following property will be very useful.

► **Lemma 7** ([8]). *Let  $M$  be any popular matching in  $G$  and let  $\vec{\alpha}$  be any witness of  $M$ . In any connected component  $C$  in the popular subgraph  $G_0$ : either (i)  $\alpha_u = 0$  for all  $u \in C$  or (ii)  $\alpha_u \in \{\pm 1\}$  for all  $u \in C$ .*

**Proof.** Consider any popular edge  $(a, b)$ . So there is some popular matching  $N$  that contains  $(a, b)$ . Since  $\text{wt}_M(\vec{N}) = \Delta(N, M) = 0$  (because  $M$  and  $N$  are popular matchings),  $\vec{N}$  is an optimal solution to (LP1). We know that  $\vec{\alpha}$  is an optimal solution to (LP2). So it follows from complementary slackness that  $\alpha_a + \alpha_b = \text{wt}_M(a, b)$ . Since  $\text{wt}_M(a, b) \in \{\pm 2, 0\}$  (an even number), the integers  $\alpha_a$  and  $\alpha_b$  have the same parity.

Let  $u$  and  $v$  be any 2 vertices in the same connected component in the popular subgraph  $G_0$ . So there is a  $u$ - $v$  path  $\rho$  in  $G$  such that every edge in  $\rho$  is a popular edge. We have just seen that the endpoints of each popular edge have the same parity in  $\vec{\alpha}$ . Hence  $\alpha_u$  and  $\alpha_v$  have the same parity. Thus either  $\alpha_u = 0$  for all  $u \in C$  or  $\alpha_u \in \{\pm 1\}$  for all  $u \in C$ . ◀

### 3 A fast exponential time algorithm for min-cost popular matching

Let  $C_1, \dots, C_r$  be the connected components in the popular subgraph  $G_0$ . Assume  $C_1, \dots, C_q$  are the non-trivial components, i.e.,  $|C_i| \geq 2$  for  $1 \leq i \leq q$  and  $|C_i| = 1$  for  $q+1 \leq i \leq r$ . So each of  $C_{q+1}, \dots, C_r$  consists of a single vertex that is left unmatched in all popular matchings in  $G$ . Call such a vertex *unpopular*. Let  $U$  be the set of unpopular vertices. The following two observations will be useful.

► **Observation 1.** *Let  $M$  be a popular matching with  $\vec{\alpha}$  as a witness. If  $u \in U$  then  $\alpha_u = 0$ .*

Proof. Since  $M$  leaves  $u$  unmatched, the self-loop  $(u, u) \in \tilde{M}$ . Observe that  $\tilde{M}$  is an optimal solution to (LP1) and  $\vec{\alpha}$  is an optimal solution to (LP2). So  $\alpha_u = \text{wt}_M(u, u) = 0$  by complementary slackness.  $\triangleleft$

► **Observation 2.** *Every non-trivial component  $C$  in the popular subgraph  $G_0$  has an even number of vertices.*

Proof. All max-size popular matchings in  $G$  leave the same vertices unmatched and these unmatched vertices are unpopular [18]. Thus a max-size popular matching  $M$  restricted to every non-trivial component  $C$  in  $G_0$  is *perfect*, i.e., all vertices in  $C$  are matched in  $M$ . Hence  $|C|$  is even.  $\triangleleft$

Let  $C_1, \dots, C_p$  be the components in  $G_0$  of size greater than 2. This means  $|C_i| \geq 4$  for  $i \in [p]$  (by Observation 2). So  $C_{p+1}, \dots, C_q$  are the components in  $G_0$  of size exactly 2.

For every subset  $I \subseteq \{1, \dots, p\}$ , our algorithm builds a corresponding graph  $G_I$ . Among all stable matchings in  $G_I$  that satisfy certain constraints, our algorithm finds a min-cost matching (call it  $N_I$ ). It will be shown that among all subsets  $I \subseteq [p]$ , the matching  $N_I$  with the least cost will map to a min-cost popular matching in  $G$ .

**The new instance  $G_I$ .** Let  $I \subseteq [p]$ . Partition the vertices in  $A \cup B$  into three subsets:

$$S_0 = \cup_{i \in I} C_i \cup U, \quad S_1 = \cup_{i \in [p] \setminus I} C_i, \quad \text{and} \quad S_2 = \cup_{i=p+1}^q C_i.$$

Our goal is to build  $G_I$  such that all popular matchings in  $G$  that admit witnesses  $\vec{\alpha}$  where  $\alpha_u = 0$  for  $u \in S_0$  and  $\alpha_u \in \{\pm 1\}$  for  $u \in S_1$  become *stable matchings* in  $G_I$ . For vertices in  $S_2$ , we do not a priori commit any particular  $\alpha$ -value. This is reflected in the vertex set  $V_I$ :

$$V_I = \{u_0 : u \in S_0 \cup S_2\} \cup \{u_+, u_-, d(u) : u \in S_1 \cup S_2\} \cup \{d'(u) : u \in S_2\}.$$

The set  $V_I$  contains a single vertex  $u_0$  for every  $u \in S_0$ , three vertices  $u_+, u_-, d(u)$  for every  $u \in S_1$ , and five vertices  $u_+, u_-, u_0, d(u), d'(u)$  for every  $u \in S_2$ . Since the  $\alpha$ -value of every  $u \in S_0$  is fixed to be 0, we have a unique vertex  $u_0$  in  $G_I$  for each  $u \in S_0$ .

Since the  $\alpha$ -value of every  $u \in S_1$  is either 1 or  $-1$ , there are two vertices  $u_+, u_-$  in  $G_I$  for each  $u \in S_1$ . However in order to map stable matchings in  $G_I$  to matchings in  $G$ , we want at most one of  $u_+, u_-$  to be matched in any stable matching in  $G_I$ : this is achieved by using a *dummy vertex*  $d(u)$ . Preferences will be such that one of  $u_+, u_-$  has to be matched to  $d(u)$  in any stable matching in  $G_I$ . So every stable matching in  $G_I$  matches at most one of  $u_+, u_-$  to a non-dummy neighbor.

Since the  $\alpha$ -value of every  $u \in S_2$  is one of  $0, \pm 1$ , we have *three* vertices  $u_+, u_-, u_0$  in  $G_I$  for each  $u \in S_2$ . However we want at most one of  $u_+, u_-, u_0$  to be matched in any stable matching in  $G_I$  and this is achieved by using *two* dummy vertices  $d(u)$  and  $d'(u)$ .

Preferences will be such that two of  $u_+, u_-, u_0$  have to be matched to  $d(u)$  and  $d'(u)$  in any stable matching in  $G_I$ . So every stable matching in  $G_I$  matches at most one of  $u_+, u_-, u_0$  to a non-dummy neighbor.

The edge set  $E_I$  of the instance  $G_I$  is defined as follows. For every  $(u, v) \in E$ , the edge set  $E_I$  consists of one or more of the following edges: (i)  $(u_0, v_0)$ , (ii)  $(u_+, v_0)$ , (iii)  $(u_0, v_+)$ , (iv)  $(u_-, v_+)$ , (v)  $(u_+, v_-)$ .

In more detail, let  $u \in A \cup B$ . Let  $v$  be a neighbor of  $u$  in  $G$ .

- if  $u, v \in S_0$  then  $(u_0, v_0)$  is in  $E_I$ .
- if  $u \in S_1$  and  $v \in S_0$  then  $(u_+, v_0)$  is in  $E_I$ .
- if  $u, v \in S_1$  and  $u$  prefers  $v$  to every neighbor in  $S_0$  then  $(u_-, v_+)$  is in  $E_I$ .

The edges in  $G_I$  that correspond to edges  $(u, v)$  in  $G$  with an endpoint, say  $u \in A \cup B$ , in  $S_2$  are described below.

- let  $v \in S_0$ . If  $u$  prefers  $v$  to its “popular partner”<sup>1</sup> then the edge  $(u_+, v_0) \in E_I$ ; else the edge  $(u_0, v_0) \in E_I$ .
- let  $v \in S_1$ . If  $u$  prefers  $v$  to its popular partner then the edge  $(u_0, v_+) \in E_I$ . If  $v$  prefers  $u$  to every neighbor in  $S_0$  then the edge  $(u_+, v_-) \in E_I$ .
- let  $v \in S_2$ . If either  $v$  is  $u$ 's popular partner or one of  $u, v$  prefers the other to its popular partner<sup>2</sup> then the edge  $(u_0, v_0) \in E_I$ . Moreover, if  $u$  prefers  $v$  to every neighbor in  $S_0$  then the edge  $(u_-, v_+) \in E_I$ .

For every  $u \in S_1$ : the edges  $(u_+, d(u))$  and  $(u_-, d(u))$  are in  $E_I$ . For every  $u \in S_2$ : the edges  $(u_+, d(u))$ ,  $(u_0, d(u))$  and the edges  $(u_0, d'(u))$ ,  $(u_-, d'(u))$  are in  $E_I$ .

**Vertex preferences.** We will first list preference orders for dummy vertices.

- For  $u \in S_1$ :  $d(u)$ 's preference order is  $u_+ \succ u_-$ , i.e., top choice  $u_+$  followed by  $u_-$ .
- For  $u \in S_2$ :  $d(u)$ 's preference order is  $u_+ \succ u_0$  and  $d'(u)$ 's preference order is  $u_0 \succ u_-$ .

Let  $u \in A \cup B$ . We now list preference orders for  $u_+, u_0$ , and  $u_-$ . An observation that will be useful here is that for any two adjacent vertices  $u, v$  in  $G$ , there is at most one element in  $\{v_0, v_+, v_-\}$  in the preference list of  $u_+$ ; similarly, in the preference lists of  $u_0$  and  $u_-$ .

1. For  $u \in S_0$ :  $u_0$ 's preference order among its neighbors in  $G_I$  is as per  $u$ 's preference order in  $G$ , i.e., ignore subscripts of vertices and arrange them as per  $u$ 's preference order in  $G$ .
2. For  $u \in S_1 \cup S_2$ :  $u_+$ 's preference order among its neighbors in  $G_I$  is as per  $u$ 's preference order in  $G$  with  $d(u)$  as its least preferred neighbor.
3. For  $u \in S_1$  (resp.,  $u \in S_2$ ):  $u_-$ 's preference order among its neighbors in  $G_I$  is  $d(u)$  (resp.,  $d'(u)$ ) as its top choice neighbor followed by its other neighbors in  $G_I$  as per  $u$ 's preference order in  $G$ .
4. For  $u \in S_2$ :  $u_0$ 's order among its neighbors in  $G_I$  is  $d(u)$  as its top choice neighbor followed by its other neighbors in  $G_I$  as per  $u$ 's preference order in  $G$  and  $d'(u)$  as its least preferred neighbor.

For  $(a, b) \in E$  and  $x, x' \in \{0, \pm\}$ , for every  $(a_x, b_{x'}) \in E_I$ , we set  $\text{cost}(a_x, b_{x'}) = \text{cost}(a, b)$ . Also, the cost of any edge incident to a dummy vertex is 0.

<sup>1</sup>  $u \in S_2$ : so  $u \in C_j$  where  $|C_j| = 2$ ; hence all popular matchings in  $G$  match  $u$  to the same neighbor.

<sup>2</sup> Note that both  $u$  and  $v$  cannot prefer each other to their respective popular partners since that would make  $(u, v)$  a blocking edge to every stable matching in  $G$ .

► **Theorem 8.** *Let  $M$  be a popular matching in  $G = (A \cup B, E)$  with a witness  $\vec{\alpha} \in \{0, \pm 1\}^n$  where  $\alpha_v = 0$  for  $v \in S_0$  and  $\alpha_v \in \{\pm 1\}$  for  $v \in S_1$ . Then there exists a stable matching  $N_I$  in  $G_I$  such that  $\text{cost}(N_I) = \text{cost}(M)$  and the following three properties are satisfied:*

1.  $N_I$  avoids all edges between a subscript + vertex and a subscript 0 vertex,
2.  $N_I$  matches all subscript – vertices, and
3.  $N_I$  includes  $q - p$  edges from the set  $\cup_{i=p+1}^q \{(a_+, b_-), (a_0, b_0), (a_-, b_+) : a, b \in C_i\}$ .

**Proof.**  $M$  is a popular matching in  $G = (A \cup B, E)$  with a witness  $\vec{\alpha} \in \{0, \pm 1\}^n$ . For any  $u \in A \cup B$ , we will define  $s_u = +/ -/ 0$  corresponding to  $\alpha_u = +1/ -1/ 0$ , respectively. That is, (i)  $\alpha_u = 1$  implies  $s_u = +$ , (ii)  $\alpha_u = -1$  implies  $s_u = -$ , and (iii)  $\alpha_u = 0$  implies  $s_u = 0$ .

- For  $u \in S_1$ : if  $s_u = +$  then let  $t_u = -$  else let  $t_u = +$ .
- For  $u \in S_2$ : if  $s_u = +$  then let  $t_u = 0$  and  $t'_u = -$ ; if  $s_u = 0$  then let  $t_u = +$  and  $t'_u = -$ ; if  $s_u = -$  then let  $t_u = +$  and  $t'_u = 0$ .

Define the set  $N_I$  as follows:

$$N_I = \{(a_{s_a}, b_{s_b}) : (a, b) \in M\} \cup \{(u_{t_u}, d(u)) : u \in S_1 \cup S_2\} \cup \{(u_{t'_u}, d'(u)) : u \in S_2\}.$$

We need to show that  $N_I \subseteq E_I$ , i.e., for every  $(a, b) \in M$ , the edge  $(a_{s_a}, b_{s_b})$  is present in  $G_I$ . Observe that  $\vec{M}$  and  $\vec{\alpha}$  are optimal solutions of (LP1) and (LP2), respectively. It follows from complementary slackness that  $\alpha_a + \alpha_b = \text{wt}_M(a, b) = 0$  for every  $(a, b) \in M$ . Thus either  $\alpha_a = \alpha_b = 0$  or  $\{\alpha_a, \alpha_b\} = \{-1, 1\}$ .

For every edge  $(a, b) \in M$  where  $\alpha_a = \alpha_b = 0$  (each such edge is in  $(S_0 \times S_0) \cup (S_2 \times S_2)$ ), observe that the edge  $(a_0, b_0)$  is in  $G_I$ . In particular, if  $(a, b) \in (S_2 \times S_2) \cap M$ , then we have  $C_i = \{a, b\}$  for some  $i \in \{p+1, \dots, q\}$  and we always include the edge  $(a_0, b_0)$  in  $G_I$ .

Consider an edge  $(a, b) \in M$  where  $\alpha_a$  or  $\alpha_b$  is  $-1$  (each such edge is in  $(S_1 \times S_1) \cup (S_2 \times S_2)$ ). Assume wlog that  $\alpha_a = -1$ . Since  $\vec{\alpha}$  is a witness of  $M$ , for every neighbor  $c \in S_0$  of  $a$ , we have  $\text{wt}_M(a, c) \leq \alpha_a + \alpha_c = -1 + 0 = -1$ . This means  $\text{wt}_M(a, c) = -2$ , i.e.,  $a$  prefers its partner in  $M$  (this is  $b$ ) to  $c$ . The constraint  $\text{wt}_M(a, c) = -2$  holds for every neighbor  $c$  of  $a$  that is in  $S_0$ . Hence it follows from the definition of the edge set of  $G_I$  that  $(a_-, b_+)$  is in  $G_I$ .

Thus every edge of  $N_I$  is present in  $G_I$ , hence  $N_I$  is a matching in  $G_I$ . We will now show that  $N_I$  obeys properties (1)-(3) given in the statement of the theorem.

1. For every edge  $(a, b) \in M$ , we have  $\alpha_a + \alpha_b = \text{wt}_M(a, b) = 0$  (by complementary slackness). Thus every edge in  $N_I$  that is not incident to any dummy vertex is of the type  $(a_+, b_-)$  or  $(a_0, b_0)$  or  $(a_-, b_+)$ . Hence  $N_I$  avoids all edges between a subscript 0 vertex and a subscript + vertex.
2. For any vertex  $u$  left unmatched in  $M$ , we have  $\alpha_u = \text{wt}_M(u, u) = 0$  (by complementary slackness). So  $u \in S_0 \cup S_2$ . Since every vertex in  $S_2$  is matched to its popular partner in all popular matchings in  $G$ , the unmatched vertex  $u \in S_0$ . Thus for every  $u \in (A \cup B) \setminus S_0$ , we have  $(u, v) \in M$  for some neighbor  $v$ : if  $\alpha_u = -1$  then  $(u_-, v_+) \in N_I$  else either  $(u_-, d(u))$  or  $(u_-, d'(u))$  is in  $N_I$ . Thus all subscript – vertices are matched in  $N_I$ .
3. For every connected component  $C_i = \{a, b\}$  in  $G_0$ , where  $p+1 \leq i \leq q$ , we know that  $(a, b) \in M$ . Thus one of  $(a_+, b_-), (a_0, b_0), (a_-, b_+)$  is in  $N_I$ . So  $N_I$  includes  $q - p$  edges from the set  $\cup_{i=p+1}^q \{(a_+, b_-), (a_0, b_0), (a_-, b_+) : a, b \in C_i\}$ .

We will now show that  $N_I$  is a stable matching in  $G_I$ . For any  $u \in A \cup B$ , it is easy to see there is no blocking edge with a dummy vertex as an endpoint. This is because a dummy vertex has only two neighbors and when it is matched to its second choice neighbor, its top choice neighbor is matched to a more preferred neighbor.

Regarding edges in  $E_I$  that correspond to edges in  $E$ , note that  $E_I$  contains certain edges of the form  $(a_0, b_0)$ ,  $(a_+, b_0)$ ,  $(a_0, b_+)$ ,  $(a_+, b_-)$ ,  $(a_-, b_+)$  for  $(a, b) \in E$ . We now need to show that no such edge in  $E_I$  blocks  $N_I$ . Consider any  $(a, b) \in E$ .

1. *Both  $a$  and  $b$  are in  $S_0$* : so  $\alpha_a = \alpha_b = 0$ . We need to show that  $(a_0, b_0)$  is not a blocking edge to  $N_I$ . Since  $\text{wt}_M(a, b) \leq \alpha_a + \alpha_b = 0$ , either  $(a, b) \in M$  or (at least) one of  $a, b$  is matched in  $M$  to a more preferred neighbor. That is, either  $(a_0, b_0) \in N_I$  or one of  $a_0, b_0$  is matched in  $N_I$  to a more preferred neighbor. So  $(a_0, b_0)$  does not block  $N_I$ .
2. *One of  $a, b$  is in  $S_0$  and the other is in  $S_1$* : assume wlog that  $a \in S_0$  and  $b \in S_1$ . So  $\alpha_a = 0$  and  $\alpha_b \in \{\pm 1\}$ . We need to show that  $(a_0, b_+)$  is not a blocking edge to  $N_I$ . There are two subcases here: (i)  $\alpha_b = 1$  and (ii)  $\alpha_b = -1$ .

In the first subcase,  $\text{wt}_M(a, b) \leq \alpha_a + \alpha_b = 1$  which implies  $\text{wt}_M(a, b) \leq 0$ . So one of  $a, b$  is matched in  $M$  to a more preferred neighbor. So one of  $a_0, b_+$  is matched in  $N_I$  to a more preferred neighbor. Hence  $(a_0, b_+)$  does not block  $N_I$ .

In the second subcase,  $\text{wt}_M(a, b) \leq \alpha_a + \alpha_b = -1$  which implies  $\text{wt}_M(a, b) = -2$ . So *both*  $a$  and  $b$  are matched in  $M$  to more preferred neighbors. In particular,  $a_0$  is matched in  $N_I$  to a neighbor preferred to  $b_+$ . Hence  $(a_0, b_+)$  does not block  $N_I$ .

3. *Both  $a$  and  $b$  are in  $S_1$* : so  $\alpha_a, \alpha_b \in \{\pm 1\}$ . We need to show that the edges  $(a_-, b_+)$  and  $(a_+, b_-)$  (whichever of these is in  $E_I$ ) do not block  $N_I$ . If  $\alpha_a = \alpha_b = 1$  then both  $a_-$  and  $b_-$  are matched to their top choice neighbors  $d(a)$  and  $d(b)$ , respectively. So neither  $(a_-, b_+)$  nor  $(a_+, b_-)$  blocks  $N_I$ .

If  $\alpha_a = 1$  and  $\alpha_b = -1$  then  $\text{wt}_M(a, b) \leq 0$ . So either  $(a, b) \in M$  or one of  $a, b$  is matched in  $M$  to a more preferred neighbor in  $G$ . That is, either  $(a_+, b_-) \in N_I$  or one of  $a_+, b_-$  is matched in  $N_I$  to a more preferred neighbor in  $G_I$ . Moreover, the edge  $(a_-, b_+)$  cannot block  $N_I$  since  $a_-$  is matched in  $N_I$  to its top choice neighbor  $d(a)$ . The subcase when  $\alpha_a = -1$  and  $\alpha_b = 1$  is symmetric.

The last subcase is  $\alpha_a = \alpha_b = -1$ . So  $\text{wt}_M(a, b) = -2$ . Hence *both*  $a$  and  $b$  are matched in  $M$  to more preferred neighbors, i.e., both  $a_-$  and  $b_-$  are matched in  $N_I$  to neighbors preferred to  $b_+$  and  $a_+$ , respectively. So neither  $(a_-, b_+)$  nor  $(a_+, b_-)$  blocks  $N_I$ .

The proofs for the remaining three cases (when at least one of  $a, b$  is in  $S_2$ ) are given below in Claims 9-11. Thus  $N_I$  is a stable matching in  $G_I$ .  $\blacktriangleleft$

$\triangleright$  **Claim 9.** Suppose one of  $a, b$  (say,  $b$ ) is in  $S_0$  and  $a$  is in  $S_2$ . Then neither  $(a_+, b_0)$  nor  $(a_0, b_0)$  blocks  $N_I$ .

*Proof.* Since  $a \in S_2$  and  $b \in S_0$ , we have  $\alpha_a \in \{0, \pm 1\}$  and  $\alpha_b = 0$ . Suppose  $\alpha_a = -1$ . Then  $\text{wt}_M(a, b) \leq -1$ , i.e.,  $\text{wt}_M(a, b) = -2$ . So both  $a$  and  $b$  are matched in  $M$  to more preferred neighbors. Since  $M$  always matches  $a$  to its *popular partner*, it means  $a$  prefers its popular partner to  $b$ . Thus  $(a_0, b_0)$  is in  $E_I$  and  $b_0$  is matched in  $N_I$  to a neighbor preferred to  $a_0$ .

Suppose  $\alpha_a \in \{0, 1\}$ . Then  $\text{wt}_M(a, b) \leq 1$ , i.e.,  $\text{wt}_M(a, b) \leq 0$ . So one of  $a, b$  is matched in  $M$  to a more preferred neighbor. Either (i)  $(a_0, b_0)$  is in  $E_I$  and so  $a_0$  is matched in  $N_I$  to a more preferred neighbor (its popular partner or  $d(a)$ ) than  $b_0$  or (ii)  $(a_+, b_0)$  is in  $E_I$ , in which case  $a$  prefers  $b$  to its popular partner – so  $b$  has to be matched in  $M$  to a neighbor preferred to  $a$ , i.e.,  $b_0$  is matched in  $N_I$  to a neighbor preferred to  $a_+$ . Hence neither  $(a_+, b_0)$  nor  $(a_0, b_0)$  (whichever is present in  $E_I$ ) blocks  $N_I$ .  $\blacktriangleleft$

$\triangleright$  **Claim 10.** Suppose one of  $a, b$  (say,  $b$ ) is in  $S_1$  and  $a$  is in  $S_2$ . Then neither  $(a_0, b_+)$  nor  $(a_+, b_-)$  blocks  $N_I$ .

*Proof.* Since  $a \in S_2$  and  $b \in S_1$ , we have  $\alpha_a \in \{0, \pm 1\}$  and  $\alpha_b \in \{\pm 1\}$ . Suppose  $a$  prefers  $b$  to its popular partner. Then  $(a_0, b_+)$  is in  $E_I$  and also  $\text{wt}_M(a, b) \geq 0$ . If  $\alpha_a = 1$  then  $a_0$  is matched to its most preferred neighbor  $d(a)$  and so  $(a_0, b_+)$  does not block  $N_I$ . If  $\alpha_a \leq 0$

## 25:10 Min-Cost Popular Matchings

then  $\alpha_b = 1$  since  $\alpha_a + \alpha_b \geq \text{wt}_M(a, b) \geq 0$ . Also  $\text{wt}_M(a, b) \leq 1$  since  $\alpha_a + \alpha_b = 1$ , i.e.,  $\text{wt}_M(a, b) = 0$ . So  $b$  has to be matched in  $M$  to a neighbor preferred to  $a$ , i.e.,  $b_+$  has to be matched in  $N_I$  to a neighbor preferred to  $a_0$ . Hence  $(a_0, b_+)$  does not block  $N_I$ .

Suppose  $b$  prefers  $a$  to all neighbors in  $S_0$ . Then  $(a_+, b_-)$  is in  $E_I$ . If  $\alpha_b = 1$  then  $b_-$  is matched to its most preferred neighbor  $d'(b)$  in  $N_I$ . Suppose  $\alpha_b = -1$ . If  $\alpha_a \in \{0, -1\}$  then  $\text{wt}_M(a, b) \leq \alpha_a + \alpha_b \leq -1$ . So  $\text{wt}_M(a, b) = -2$ . This means both  $a, b$  are matched in  $M$  to more preferred neighbors. Hence  $b_-$  is matched in  $N_I$  to a neighbor preferred to  $a_+$ . Suppose  $\alpha_a = 1$ . Then  $\text{wt}_M(a, b) \leq 0$ : so one of  $a, b$  is matched in  $M$  to a more preferred neighbor. So one of  $a_+, b_-$  is matched in  $N_I$  to a more preferred neighbor. Thus the edge  $(a_+, b_-)$  does not block  $N_I$ .  $\triangleleft$

$\triangleright$  **Claim 11.** Suppose both  $a$  and  $b$  are in  $S_2$ . Then none of the edges  $(a_0, b_0), (a_+, b_-), (a_-, b_+)$  blocks  $N_I$ .

*Proof.* Since  $a, b$  are in  $S_2$ , we have  $\alpha_a, \alpha_b \in \{0, \pm 1\}$ . If  $a, b$  are each other's popular partners or one of them prefers the other to its popular partner then the edge  $(a_0, b_0)$  is in  $E_I$  and also  $\text{wt}_M(a, b) \geq 0$ . So either  $\alpha_a = \alpha_b = 0$  or at least one of  $\alpha_a, \alpha_b$  is 1. So either  $(a_0, b_0) \in N_I$  or one of  $a_0, b_0$  is matched in  $N_I$  to a more preferred neighbor. Thus  $(a_0, b_0)$  does not block  $N_I$ .

If  $a$  prefers  $b$  to all its neighbors in  $S_0$  then the edge  $(a_-, b_+)$  is in  $E_I$ . If  $\alpha_a \in \{0, 1\}$  then  $a_-$  is matched to its most preferred neighbor  $d'(a)$  in  $N_I$ . So the edge  $(a_-, b_+)$  does not block  $N_I$ . Suppose  $\alpha_a = -1$ . If  $\alpha_b \in \{0, -1\}$  then  $\text{wt}_M(a, b) \leq \alpha_a + \alpha_b \leq -1$ . So  $\text{wt}_M(a, b) = -2$ . This means both  $a, b$  are matched in  $M$  to more preferred neighbors. Hence  $a_-$  is matched in  $N_I$  to a neighbor preferred to  $b_+$ . Suppose  $\alpha_b = 1$ . Then  $\text{wt}_M(a, b) \leq 0$ : so one of  $a, b$  is matched in  $M$  to a more preferred neighbor. So one of  $a_-, b_+$  is matched in  $N_I$  to a more preferred neighbor. Thus the edge  $(a_-, b_+)$  does not block  $N_I$ .

The analysis that  $(a_+, b_-)$  does not block  $N_I$  when  $b$  prefers  $a$  to all neighbors in  $S_0$  is analogous.  $\triangleleft$

Let us call a stable matching in  $G_I$  that satisfies the three properties given in Theorem 8 a *desired* stable matching. Theorem 12 proves the converse of Theorem 8.

$\blacktriangleright$  **Theorem 12.** Suppose  $G_I$  admits a desired stable matching, say  $N_I$ . Then  $N_I$  can be mapped to a popular matching  $M$  in  $G$  such that  $\text{cost}(N_I) = \text{cost}(M)$ .

**Proof.** The matching  $M$  will be defined as follows:

$$M = \{(a, b) : (a_{s_a}, b_{s_b}) \in N_I \text{ for } s_a, s_b \in \{0, \pm\}\}.$$

For any  $u \in A \cup B$ , at most one of  $u_+, u_0, u_-$  can be matched to a non-dummy neighbor in  $N_I$ . Thus  $M$  is a valid matching in  $G$ . In order to prove  $M$ 's popularity, we will show a witness  $\vec{\alpha} \in \{0, \pm 1\}^n$ . Define  $\alpha_u = 0$  for all  $u \in S_0$ . Let  $u \in S_1$ . Since  $N_I$  is stable, the vertex  $d(u)$  (as the top choice neighbor of  $u_-$ ) has to be matched in  $N_I$ . So for  $u \in S_1$ , define  $\alpha_u$  as follows:

$$\text{let } \alpha_u = \begin{cases} -1 & \text{if } (u_+, d(u)) \in N_I \\ 1 & \text{if } (u_-, d(u)) \in N_I. \end{cases}$$

Let  $u \in S_2$ . Then there are two dummy vertices  $d(u)$  and  $d'(u)$  for  $u$  and both of them (as the top choice neighbors of  $u_0$  and  $u_-$ , resp.) have to be matched in  $N_I$ . So for  $u \in S_2$ , define  $\alpha_u$  as follows:

$$\text{let } \alpha_u = \begin{cases} -1 & \text{if } (u_+, d(u)) \text{ and } (u_0, d'(u)) \text{ are in } N_I \\ 0 & \text{if } (u_+, d(u)) \text{ and } (u_-, d'(u)) \text{ are in } N_I \\ 1 & \text{if } (u_0, d(u)) \text{ and } (u_-, d'(u)) \text{ are in } N_I. \end{cases}$$

We will now show that  $\vec{\alpha}$  is a witness of  $M$ 's popularity. Observe that all edges in  $N_I$  not involving any dummy vertex are of the form  $(a_+, b_-)$  or  $(a_0, b_0)$  or  $(a_-, b_+)$ . This is because  $N_I$  avoids all edges of the type  $(a_+, b_0)$  and  $(a_0, b_+)$  (by property (1)). Thus  $\alpha_a + \alpha_b = 0$  for all  $(a, b) \in M$ . Due to property (2), property (3), and  $N_I$ 's stability, it follows that for any vertex  $u$  left unmatched in  $M$ , we have  $u \in S_0$ , i.e.,  $\alpha_u = 0$ . So  $\sum_{u \in A \cup B} \alpha_u = 0$ .

It is also easy to see that  $\alpha_u \geq \text{wt}_M(u, u)$  for every vertex  $u$ . This is because every vertex  $u \in (A \cup B) \setminus S_0$  is matched in  $M$  and so we have  $\alpha_u \geq -1 = \text{wt}_M(u, u)$  for these vertices. For any vertex  $u \in S_0$ , we have  $\alpha_u = 0 \geq \text{wt}_M(u, u)$ .

What is left to show is that every edge  $(a, b)$  in  $G$  is *covered*, i.e.,  $\alpha_a + \alpha_b \geq \text{wt}_M(a, b)$ . This is proved below in Lemma 13. Thus  $\vec{\alpha}$  is a witness of  $M$  (by Theorem 5). So  $M$  is a popular matching; also  $\text{cost}(M) = \text{cost}(N_I)$ . This finishes the proof of Theorem 12.  $\blacktriangleleft$

► **Lemma 13.** *We have  $\alpha_a + \alpha_b \geq \text{wt}_M(a, b)$  for every edge  $(a, b)$  in  $G$ .*

**Proof.** Recall that  $\text{wt}_M(a, b) \in \{0, \pm 2\}$ . Any edge  $(a, b)$  where  $\alpha_a = \alpha_b = 1$  is obviously covered since  $\text{wt}_M(a, b) \leq 2$ . The proofs for other cases of  $(\alpha_a, \alpha_b)$  are given in Claims 14-18.

▷ **Claim 14.** Any edge  $(a, b)$  where  $\{\alpha_a, \alpha_b\} = \{0, 1\}$  is covered.

**Proof.** Assume without loss of generality  $\alpha_a = 1$  and  $\alpha_b = 0$ : so  $a \in S_1 \cup S_2$  and  $b \in S_0 \cup S_2$ . If the edge  $(a_+, b_0)$  is in  $E_I$  then the stability of  $N_I$  implies that either (i)  $a_+$  is matched in  $N_I$  to a neighbor preferred to  $b_0$  or (ii)  $b_0$  is matched in  $N_I$  to a neighbor preferred to  $a_+$  (moreover, a non-dummy neighbor since  $\alpha_b = 0$ ). So at least one of  $a, b$  is matched in  $M$  to a more preferred neighbor. Thus  $\text{wt}_M(a, b) \leq 0$ .

The edge  $(a_+, b_0)$  is *not* present in  $G_I$  in the following 2 cases:

1. both  $a, b$  are in  $S_2$  and either (i)  $a, b$  are each other's *popular partners* or (ii) at least one of  $a, b$  prefers its popular partner to the other (see footnote 2). By property (3), every vertex in  $S_2$  is matched in  $M$  to its popular partner. Hence  $\text{wt}_M(a, b) \leq 0$ .
2. either (i)  $a \in S_2$  prefers its popular partner (call it  $y$ ) to  $b \in S_0$  or (ii)  $b \in S_2$  prefers its popular partner (call it  $z$ ) to  $a \in S_1$ ; property (3) forces  $(a, y)$  to be in  $M$  in the first case and  $(z, b)$  to be in  $M$  in the second case. So  $\text{wt}_M(a, b) \leq 0$ .

Hence in all cases, we have  $\text{wt}_M(a, b) \leq 0 < 1 = \alpha_a + \alpha_b$ .  $\blacktriangleleft$

▷ **Claim 15.** Any edge  $(a, b)$  where  $\alpha_a = \alpha_b = 0$  is covered.

**Proof.** Since  $\alpha_a = \alpha_b = 0$ , we have  $a, b \in S_0 \cup S_2$ . If the edge  $(a_0, b_0)$  is in  $G_I$ , then it follows from the stability of  $N_I$  that  $(a_0, b_0) \in N_I$  or one of  $a_0, b_0$  is matched in  $N_I$  to a more preferred (non-dummy) neighbor, i.e., at least one of  $a, b$  is matched in  $M$  to a more preferred neighbor. Thus  $\text{wt}_M(a, b) \leq 0$ .

The edge  $(a_0, b_0)$  is *not* present in  $G_I$  in the following 2 cases:

1.  $a \in S_2$  prefers  $b \in S_0$  to its popular partner: in this case  $(a_+, b_0)$  is in  $G_I$ . Since  $\alpha_a = 0$ , the vertex  $a_+$  is matched in  $N_I$  to its least preferred neighbor  $d(a)$ . Thus it follows from the stability of  $N_I$  that  $b_0$  is matched to a more preferred neighbor than  $a_+$ , so  $\text{wt}_M(a, b) \leq 0$ . It is similar when  $b \in S_2$  prefers  $a \in S_0$  to its popular partner.
2. both  $a, b$  are in  $S_2$  and they prefer their respective popular partners to each other: in this case  $\text{wt}_M(a, b) = -2$ .

Hence in all cases, we have  $\text{wt}_M(a, b) \leq 0 = \alpha_a + \alpha_b$ .  $\blacktriangleleft$

▷ **Claim 16.** Any edge  $(a, b)$  where  $\{\alpha_a, \alpha_b\} = \{-1, 1\}$  is covered.

## 25:12 Min-Cost Popular Matchings

Proof. Assume without loss of generality that  $\alpha_a = 1$  and  $\alpha_b = -1$ . We need to show that  $\text{wt}_M(a, b) \leq 0$ . Either (i)  $(a_+, b_-) \in N_I$  or (ii)  $(a_+, y_-)$  and  $(z_+, b_-)$  are in  $N_I$  for some neighbors  $y, z$  of  $a, b$ , respectively. In case (i),  $\text{wt}_M(a, b) = 0$ . In case (ii), we will consider 2 subcases.

1. Suppose  $a, b \in S_1$  or  $a, b \in S_2$  or  $a \in S_2$  and  $b \in S_1$ . Since the edge  $(z_+, b_-)$  is in  $G_I$ ,  $b$  prefers  $z$  to all its neighbors in  $S_0$ . Hence if  $b$  prefers  $a$  to  $z$  then the edge  $(a_+, b_-)$  has to be present in  $G_I$ . It follows from the stability of  $N_I$  that  $a_+$  prefers  $y_-$  to  $b_-$ , i.e.,  $a$  prefers  $y$  to  $b$ . Hence  $\text{wt}_M(a, b) \leq 0$ .
2. The remaining case is when  $a \in S_1$  and  $b \in S_2$ . So  $z$  is  $b$ 's popular partner. If  $b$  prefers  $a$  to  $z$  then the edge  $(a_+, b_0)$  is present in  $G_I$ . Since  $b_0$  is matched to its least preferred neighbor  $d'(b)$  in  $N_I$ , the stability of  $N_I$  implies that  $a_+$  prefers  $y_-$  to  $b_0$ , i.e.,  $a$  prefers  $y$  to  $b$ . Hence  $\text{wt}_M(a, b) \leq 0$ .

Hence in all cases, we have  $\text{wt}_M(a, b) \leq 0 = \alpha_a + \alpha_b$ .  $\triangleleft$

$\triangleright$  Claim 17. Any edge  $(a, b)$  where  $\alpha_a = \alpha_b = -1$  is covered.

Proof. So  $(a_-, y_+)$  and  $(z_+, b_-)$  are in  $N_I$  for some neighbors  $y, z$  of  $a, b$ , respectively. There are 3 cases here:

1. Both  $a$  and  $b$  are in  $S_1$ . Suppose  $a$  prefers  $b$  to  $y$ . Then the edge  $(a_-, b_+)$  is present in  $G_I$  since  $a$  prefers  $y$  (and thus  $b$ ) to all neighbors in  $S_0$ ; moreover,  $b_+$  prefers  $a_-$  to  $d(b)$ . Hence  $(a_-, b_+)$  would be a blocking edge to  $N_I$ , contradicting its stability. So  $a$  prefers  $y$  to  $b$ . Similarly,  $b$  prefers  $z$  to  $a$ . Thus  $\text{wt}_M(a, b) = -2$ .
2. Both  $a$  and  $b$  are in  $S_2$ . Either both  $a$  and  $b$  prefer their popular partners ( $y$  and  $z$ , resp.) to each other or the edge  $(a_0, b_0)$  is in  $G_I$ . In the latter case,  $(a_0, b_0)$  would be blocking edge to  $N_I$  since  $N_I$  contains  $(a_0, d'(a))$  and  $(b_0, d'(b))$ . Thus both  $a$  and  $b$  prefer their popular partners to each other, so  $\text{wt}_M(a, b) = -2$ .
3. One of  $a, b$  is in  $S_2$  and the other is in  $S_1$ : assume wlog that  $a \in S_2$  and  $b \in S_1$ . We claim that  $b$  prefers  $z$  to  $a$ . Otherwise the edge  $(a_+, b_-)$  would be in  $G_I$  since  $b$  prefers  $z$  (and thus  $a$ ) to all neighbors in  $S_0$ . Note that  $(a_+, b_-)$  would block  $N_I$  since  $(a_+, d(a)) \in N_I$ . We next claim that  $a$  prefers  $y$  to  $b$ . Otherwise the edge  $(a_0, b_+)$  would be in  $G_I$  and this would be a blocking edge to  $N_I$  since  $(a_0, d'(a))$  and  $(b_+, d(b))$  are in  $N_I$ . Thus both  $a$  and  $b$  prefer their partners in  $M$  to each other, so  $\text{wt}_M(a, b) = -2$ .

Hence in all cases, we have  $\text{wt}_M(a, b) = -2 = \alpha_a + \alpha_b$ .  $\triangleleft$

$\triangleright$  Claim 18. Any edge  $(a, b)$  where  $\{\alpha_a, \alpha_b\} = \{-1, 0\}$  is covered.

Proof. Assume wlog  $\alpha_a = -1$  and  $\alpha_b = 0$ . So  $(a_-, y_+) \in N_I$  for some neighbor  $y$  of  $a$ . Also  $(a_+, d(a)) \in N_I$ . Observe that  $b_0$  has to be matched in  $N_I$ , otherwise one of  $(a_+, b_0), (a_0, b_0)$  – whichever is present in  $G_I$  – would be a blocking edge to  $N_I$ . So  $(z_0, b_0)$  is in  $N_I$  for some neighbor  $z$  of  $b$ .

If the edge  $(a_+, b_0)$  is present in  $E_I$  then it follows from the stability of  $N_I$  that  $b_0$  prefers  $z_0$  to  $a_+$ , i.e.,  $b$  prefers  $z$  to  $a$ . Moreover, it follows from the existence of the edge  $(a_-, y_+)$  in  $E_I$  that  $a$  prefers  $y$  to all its neighbors in  $S_0$ , i.e.,  $a$  prefers  $y$  to  $b$  if  $b \in S_0$ . If  $b \in S_2$  and  $a$  prefers  $b$  to  $y$  then  $a$  prefers  $b$  to all neighbors in  $S_0$  and so the edge  $(a_-, b_+)$  would have been present in  $E_I$ . This would have been a blocking edge to  $N_I$  since  $a_-$  prefers  $b_+$  to  $y_+$  and  $b_+$  prefers  $a_-$  to  $d(b)$ . Thus  $a$  prefers  $y$  to  $b$  and so  $\text{wt}_M(a, b) = -2$ .

The cases when  $(a_+, b_0)$  is *not* present in  $G_I$  are the following:

1. Both  $a, b$  are in  $S_2$ : there are two subcases here. In the first subcase, both  $a$  and  $b$  prefer their popular partners to each other and so  $\text{wt}_M(a, b) = -2$ . In the second subcase, one of  $a, b$  prefers the other to its popular partner. Then the edge  $(a_0, b_0)$  is in  $E_I$  and the

stability of  $N_I$  implies that  $b_0$  prefers  $z_0$  to  $a_0$  since  $(a_0, d'(a)) \in N_I$ . Thus  $b$  prefers  $z$  to  $a$ . This means that  $a$  prefers  $b$  to  $y$  and the edge  $(a_-, b_+)$  has to be in  $E_I$  since  $a$  prefers  $y$  (and thus  $b$ ) to all neighbors in  $S_0$ . This makes  $(a_-, b_+)$  a blocking edge to  $N_I$ , a contradiction. Hence both  $a, b$  prefer their popular partners to each other, i.e., the second subcase does not arise. Thus  $\text{wt}_M(a, b) = -2$ .

2.  $b \in S_2$  prefers its popular partner to  $a \in S_1$ : so  $b$  prefers  $z$  to  $a$  and we have to argue that  $a$  prefers  $y$  to  $b$ . Suppose not, i.e.,  $a$  prefers  $b$  to  $y$ . Since the edge  $(a_-, y_+)$  is in  $E_I$ ,  $a$  prefers  $y$  (and thus  $b$ ) to all neighbors in  $S_0$ . So the edge  $(a_-, b_+)$  is in  $E_I$  and this is a blocking edge to  $N_I$  since  $(b_+, d(b))$  and  $(a_-, y_+)$  are in  $N_I$ . This contradicts  $N_I$ 's stability, hence  $a$  prefers  $y$  to  $b$ . Thus  $\text{wt}_M(a, b) = -2$ .
3.  $a \in S_2$  prefers its popular partner to  $b \in S_0$ : so  $a$  prefers  $y$  to  $b$ . Then the edge  $(a_0, b_0)$  is in  $G_I$ . Since  $a_0$  is matched to its least preferred neighbor  $d'(a)$ , it follows from the stability of  $N_I$  that  $b_0$  prefers  $z_0$  to  $a_0$ , i.e.,  $b$  prefers  $z$  to  $a$ . Thus  $\text{wt}_M(a, b) = -2$ .

Hence in all cases, we have  $\text{wt}_M(a, b) = -2 < -1 = \alpha_a + \alpha_b$ .  $\triangleleft$

This finishes the proof of Lemma 13.  $\blacktriangleleft$

**Finding a min-cost desired stable matching in  $G_I$ .** We first check that all subscript  $-$  vertices are stable in  $G_I$ . This is easily done by running Gale-Shapley algorithm in  $G_I$  and using the fact that all stable matchings leave the same vertices unmatched [15]. This ensures property (2). Then we solve a min-cost stable matching problem in  $G_I$  with *forbidden* edges. There are two types of forbidden edges here: the first type are all edges between a subscript  $+$  vertex and a subscript  $0$  vertex in  $G_I$ . Forbidding these edges ensures property (1). The second type of forbidden edges are described below. Forbidding these edges ensures property (3).

**Ensuring property (3).** For any  $u \in S_2$ , all edges incident to any vertex  $u_+, u_0, u_-$  are marked forbidden *except* for the following edges, where  $v$  is  $u$ 's popular partner:

- the edges among  $(u_+, v_-), (u_0, v_0), (u_-, v_+)$  that are in  $E_I$ ;
- the pair of edges  $(u_+, d(u)), (u_0, d(u))$  and the pair of edges  $(u_0, d'(u)), (u_-, d'(u))$ .

For  $u \in S_2$ , every stable matching in  $G_I$  has to match  $u_+, u_0, d(u), d'(u)$  since these are top choice neighbors for some vertices. Moreover, we have already checked that all subscript  $-$  vertices are stable in  $G_I$ . Thus all the five vertices  $u_+, u_0, u_-, d(u), d'(u)$  have to be matched in every stable matching in  $G_I$ . In particular, two of  $u_+, u_0, u_-$  are matched to  $d(u), d'(u)$ . Thus any stable matching in  $G_I$  that avoids forbidden edges of the second type has to contain one of  $(u_+, v_-), (u_0, v_0), (u_-, v_+)$ .

**Desired stable matchings.** We have seen that all stable matchings of  $G_I$  that satisfy the 3 properties given in Theorem 8 are precisely those stable matchings in  $G_I$  that avoid edges that we marked forbidden. Consider the stable matching polytope  $\mathcal{S}$  of  $G_I$ : we know that  $x_e \geq 0$  for any edge  $e$  is a valid inequality for  $\mathcal{S}$ , hence the intersection of  $\mathcal{S}$  with the constraints  $x_e = 0$  for every forbidden edge  $e$  is a face  $F$  of  $\mathcal{S}$ . Since  $F$  is an integral polytope and every integral point in  $F$  is a stable matching in  $G_I$  that avoids forbidden edges,  $N_I$  can be computed in polynomial time by linear programming over the constraints defining  $F$ . These are the constraints of the stable matching polytope  $\mathcal{S}$  along with the constraints  $x_e = 0$  for every forbidden edge  $e$ . A min-cost desired stable matching  $N_I$  over all  $I \subseteq [p]$  maps to a min-cost popular matching in  $G$  (by Theorem 8 and Theorem 12).

As mentioned earlier, the popular subgraph  $G_0$  can be constructed in linear time [7]. Then we identify the connected components  $C_1, \dots, C_p$  of size at least 4 in  $G_0$ . The number of sets  $I$  that we need to go through is  $2^p$ , thus our algorithm runs in  $2^p \cdot \text{poly}(n)$  time. Since  $p \leq n/4$ , this proves Theorem 2 stated in Section 1.

#### 4 Semi-popular matchings

In this section we consider the problem of computing an *almost* semi-popular matching of cost at most  $2\text{opt}$ . Our input is a bipartite graph  $G = (A \cup B, E)$  where vertices have strict preferences and we have  $\text{cost} : E \rightarrow \mathbb{R}_{\geq 0}$ . We are also given a parameter  $\varepsilon \in (0, 1/2)$ .

**Popular fractional matchings.** The notion of popularity can be extended to fractional matchings. A vector  $\vec{x} \in \mathbb{R}_{\geq 0}^{|E|}$  that satisfies  $\sum_{e \in \delta(u)} x_e \leq 1$  for all vertices  $u$  is a fractional matching in  $G$ . The fractional matching  $\vec{x}$  is popular if  $\Delta(\vec{x}, N) \geq 0$  for all matchings  $N$ , where  $\Delta(\vec{x}, N)$  is defined as follows:  $\vec{x}$  is a convex combination of matchings (Birkhoff-von Neumann theorem), so  $\vec{x} = \sum_i p_i I_{M_i}$  for some matchings  $M_i$  where  $\sum_i p_i = 1$ , each  $p_i \geq 0$ , and  $\Delta(\vec{x}, N)$  is defined as  $\sum_i p_i \cdot \Delta(M_i, N)$ . Since the fractional matching  $\vec{x}$  can possibly be expressed in multiple ways as convex combinations of matchings,  $\Delta(\vec{x}, N)$  may seem ill-defined. However this is well-defined and we refer to [26, Lemma 1] for details.

Let  $\text{opt}^*$  be the cost of a min-cost popular fractional matching in  $G$  and let  $\vec{q}$  be a min-cost popular fractional matching. The fractional matching  $\vec{q}$  can be efficiently computed [26]. We have  $\text{cost}(\vec{q}) = \text{opt}^* \leq \text{opt}$  where  $\text{opt}$  is the cost of a min-cost popular matching.

It was shown in [20] that the popular fractional matching polytope is half-integral. Thus we can assume that  $\vec{q}$  is half-integral. So  $\vec{q} = (I_{M_1} + I_{M_2})/2$  where  $M_1$  and  $M_2$  are two matchings in  $G$ . We know that  $\Delta(\vec{q}, N) \geq 0$  for all matchings  $N$  in  $G$ .

► **Observation 3.** *There is a matching  $M \in \{M_1, M_2\}$  such that  $M$  is semi-popular.*

Proof. Since  $\Delta(\vec{q}, N) = (\Delta(M_1, N) + \Delta(M_2, N))/2$  and  $\Delta(\vec{q}, N) \geq 0$  for every matching  $N$ , we have either  $\Delta(M_1, N) \geq 0$  or  $\Delta(M_2, N) \geq 0$  for every matching  $N$ . Hence one of  $M_1, M_2$  is undefeated by at least half the matchings in  $G$ . ◀

Since all edge costs are non-negative and  $\text{cost}(\vec{q}) = (\text{cost}(M_1) + \text{cost}(M_2))/2$ , we have  $\text{cost}(M_1) \leq 2 \cdot \text{cost}(\vec{q})$  and  $\text{cost}(M_2) \leq 2 \cdot \text{cost}(\vec{q})$ . So there is  $M \in \{M_1, M_2\}$  such that (i)  $M$  is semi-popular and (ii)  $\text{cost}(M) \leq 2\text{opt}$ .

The problem here is to efficiently decide which of  $M_1, M_2$  is semi-popular. We do not know how to answer this question exactly. However we can decide with high probability whether both  $M_1$  and  $M_2$  are close to being semi-popular or one of them is not - in which case the other matching has to be semi-popular (by Observation 3). Here we will use the classical result from [22] that shows a polynomial time algorithm to sample matchings from a distribution that is close to the uniform distribution in total variation distance (see [22, Corollary 4.3]).

The input is  $G = (A \cup B, E)$  with non-negative edge costs and  $\varepsilon \in (0, 1/2)$ . Our algorithm is as follows:

1. Compute a min-cost popular half-integral matching  $\vec{q}$  in  $G$ . Let  $\vec{q} = (I_{M_1} + I_{M_2})/2$  where  $M_1$  and  $M_2$  are matchings in  $G$ .
2. Produce a sample  $\mathcal{S}$  of  $s = 64 \cdot \lceil (\ln n)/\varepsilon^2 \rceil$  matchings from a distribution that is  $\varepsilon/4$ -close to the uniform distribution (on all matchings in  $G$ ) in total variation distance.
3. If both  $M_1$  and  $M_2$  are undefeated by more than  $s \cdot (1 - \varepsilon)/2$  of matchings in  $\mathcal{S}$  then return the matching in  $\{M_1, M_2\}$  with lower cost.
4. Else return the matching in  $\{M_1, M_2\}$  undefeated by a majority of matchings in  $\mathcal{S}$ .

In Step 2, we use the random sampler in [22] that constructs the sample  $\mathcal{S}$  in  $\text{poly}(n, \frac{1}{\varepsilon})$  time. It is easy to see that the running time of our algorithm is  $\text{poly}(n, \frac{1}{\varepsilon})$ . Lemma 19 and Lemma 20 bound the probability that our algorithm makes an error.

► **Lemma 19.** *Suppose  $M \in \{M_1, M_2\}$  is defeated by more than  $1/2 + \varepsilon$  fraction of all matchings in  $G$ . Then our algorithm returns  $M$  in step 3 with probability at most  $1/n$ .*

**Proof.** Since  $M$  is defeated by more than  $1/2 + \varepsilon$  fraction of all matchings in  $G$ , the expected number of matchings that defeat  $M$  from a set of  $s$  matchings, where each matching is chosen uniformly at random from the set of all matchings in  $G$  is more than  $s \cdot (1/2 + \varepsilon)$ . The set  $\mathcal{S}$  is formed by sampling  $s$  matchings from a distribution  $\varepsilon/4$ -close to the uniform distribution in total variation distance. Hence the expected number of matchings from  $\mathcal{S}$  that defeat  $M$  is more than  $s \cdot (1/2 + \varepsilon - \varepsilon/4) = s \cdot (2 + 3\varepsilon)/4$ .

If  $M$  was returned in step 3 then  $M$  was undefeated by more than  $s \cdot (1 - \varepsilon)/2$  matchings from  $\mathcal{S}$ . Equivalently, less than  $s \cdot (1 + \varepsilon)/2$  matchings from  $\mathcal{S}$  defeated  $M$ . By Chernoff bound, the probability of this event is at most  $\exp(-s \cdot \varepsilon^2 / (16 + 24\varepsilon))$ . Since  $s \geq 64 \cdot (\ln n) / \varepsilon^2$ , this probability is at most  $1/n$ . ◀

The next lemma bounds the error when our algorithm reaches step 4.

► **Lemma 20.** *Suppose  $M \in \{M_1, M_2\}$  is not semi-popular. Then our algorithm returns  $M$  in step 4 with probability at most  $1/n$ .*

**Proof.** Since  $M$  is defeated by more than half the matchings in  $G$ , the expected number of matchings that defeat  $M$  from a set of  $s$  matchings, where each matching is chosen uniformly at random from the set of all matchings in  $G$ , is more than  $s/2$ . Since the set  $\mathcal{S}$  is formed by sampling  $s$  matchings from a distribution  $\varepsilon/4$ -close to the uniform distribution in total variation distance, the expected number of matchings that defeat  $N$  from  $\mathcal{S}$  is more than  $s \cdot (2 - \varepsilon)/4$ .

The algorithm reached step 4 and  $M$  was the matching that was undefeated by a majority of matchings in  $\mathcal{S}$ . Observe that  $M$  defeated more than  $s \cdot (1 + \varepsilon)/2$  matchings in the set  $\mathcal{S}$ . This is because the matching in  $\{M_1, M_2\} \setminus \{M\}$  was defeated by more than  $s \cdot (1 + \varepsilon)/2$  matchings in  $\mathcal{S}$  – otherwise we would not have reached step 4. Since  $M$  defeats more than  $s(1 + \varepsilon)/2$  matchings from  $\mathcal{S}$ , less than  $s(1 - \varepsilon)/2$  matchings from  $\mathcal{S}$  defeated  $M$ . By Chernoff bound, the probability of this event is at most  $\exp(-s \cdot \varepsilon^2 / (64 - 32\varepsilon))$ . Since  $s \geq 64 \cdot (\ln n) / \varepsilon^2$ , this probability is at most  $1/n$ . ◀

Lemma 19 and Lemma 20 bound the error probability of our algorithm. Thus we have proved Theorem 4 stated in Section 1.

---

## References

- 1 D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.
- 2 P. Biro, R. W. Irving, and D. F. Manlove. Popular matchings in the marriage and roommates problems. In *Proceedings of the seventh International Conference on Algorithms and Complexity (CIAC)*, pages 97–108, 2010.
- 3 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'Imprimerie Royale, 1785.
- 4 Condorcet method. [https://en.wikipedia.org/wiki/Condorcet\\_method](https://en.wikipedia.org/wiki/Condorcet_method).
- 5 Á. Cseh. Popular matchings. Trends in Computational Social Choice, Ulle Endriss (ed.), 2017.

- 6 Á. Cseh, C.-C. Huang, and T. Kavitha. Popular matchings with two-sided preferences and one-sided ties. *SIAM Journal on Discrete Mathematics*, 31(4):2348–2377, 2017.
- 7 Á. Cseh and T. Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.
- 8 Y. Faenza and T. Kavitha. Quasi-popular matchings, optimality, and extended formulations. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 325–344, 2020.
- 9 Y. Faenza, T. Kavitha, V. Powers, and X. Zhang. Popular matchings and limits to tractability. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2790–2809, 2019.
- 10 T. Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45(2):233–284, 1992.
- 11 T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11(3):291–319, 1994.
- 12 T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.
- 13 F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer-Verlag New York, Inc., New York, 2010.
- 14 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
- 15 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.
- 16 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.
- 17 S. Gupta, P. Misra, S. Saurabh, and M. Zehavi. Popular matching in roommates setting is np-hard. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2810–2822, 2019.
- 18 M. Hirakawa, Y. Yamauchi, S. Kijima, and M. Yamashita. On the structure of popular matchings in the stable marriage problem - who can join a popular matching? In the 3rd International Workshop on Matching Under Preferences (MATCH-UP), 2015.
- 19 C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.
- 20 C.-C. Huang and T. Kavitha. Popularity, mixed matchings, and self-duality. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2294–2310, 2017.
- 21 R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the “optimal” stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.
- 22 M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.
- 23 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.
- 24 T. Kavitha. Popular half-integral matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22:1–22:13, 2016.
- 25 T. Kavitha. Popular roommates in simply exponential time. In *Proceedings of the 39th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 20:1–20:15, 2019.
- 26 T. Kavitha, J. Mestre, and M. Nasre. Popular mixed matchings. *Theoretical Computer Science*, 412:2679–2690, 2011.
- 27 K. Makarychev, Y. Makarychev, M. Sviridenko, and J. Ward. A bi-criteria approximation algorithm for  $k$ -means. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 14:1–14:20, 2016.
- 28 E. McDermid and R. W. Irving. Sex-equal stable matchings: Complexity and exact algorithms. *Algorithmica*, 68(3):545–570, 2014.

- 29 U. G. Rothblum. Characterization of stable matchings as extreme points of a polytope. *Mathematical Programming*, 54:57–67, 1992.
- 30 C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.
- 31 J. H. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 8(3):147–153, 1989.



# Constructing Large Matchings via Query Access to a Maximal Matching Oracle

Lidiya Khalidah binti Khalil

Department of Computer Science, University of Bristol, UK  
lb17727@bristol.ac.uk

Christian Konrad<sup>1</sup> 

Department of Computer Science, University of Bristol, UK  
<http://people.cs.bris.ac.uk/~konrad/>  
christian.konrad@bristol.ac.uk

---

## Abstract

Multi-pass streaming algorithm for **Maximum Matching** have been studied since more than 15 years and various algorithmic results are known today, including 2-pass streaming algorithms that break the  $1/2$ -approximation barrier, and  $(1 - \epsilon)$ -approximation streaming algorithms that run in  $O(\text{poly } \frac{1}{\epsilon})$  passes in bipartite graphs and in  $O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}})$  or  $O(\text{poly}(\frac{1}{\epsilon}) \cdot \log n)$  passes in general graphs, where  $n$  is the number of vertices of the input graph. However, proving impossibility results for such algorithms has so far been elusive, and, for example, even the existence of 2-pass small space streaming algorithms with approximation factor 0.999 has not yet been ruled out.

The key building block of all multi-pass streaming algorithms for **Maximum Matching** is the **GREEDY** matching algorithm. Our aim is to understand the limitations of this approach: How many passes are required if the algorithm solely relies on the invocation of the **GREEDY** algorithm?

In this paper, we initiate the study of lower bounds for restricted families of multi-pass streaming algorithms for **Maximum Matching**. We focus on the simple yet powerful class of algorithms that in each pass run **GREEDY** on a vertex-induced subgraph of the input graph. In bipartite graphs, we show that 3 passes are necessary and sufficient to improve on the trivial approximation factor of  $1/2$ : We give a lower bound of 0.6 on the approximation ratio of such algorithms, which is optimal. We further show that  $\Omega(\frac{1}{\epsilon})$  passes are required for computing a  $(1 - \epsilon)$ -approximation, even in bipartite graphs. Last, the considered class of algorithms is not well-suited to general graphs: We show that  $\Omega(n)$  passes are required in order to improve on the trivial approximation factor of  $1/2$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Streaming, sublinear and near linear time algorithms; Theory of computation  $\rightarrow$  Graph algorithms analysis

**Keywords and phrases** Maximum matching approximation, Query model, Streaming algorithms

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.26

## 1 Introduction

The **GREEDY** matching algorithm is the key building block of most published streaming algorithms for approximate **Maximum Matching** [16, 26, 13, 25, 2, 14, 21, 24]. Given a graph  $G = (V, E)$ , **GREEDY** scans the set of edges  $E$  in arbitrary order and inserts the current edge  $e \in E$  into an initially empty matching  $M$  if possible, i.e., if both endpoints of  $e$  are not yet matched by an edge in  $M$ . **GREEDY** produces a maximal matching, which is known to be at least half as large as a matching of largest size.

The **GREEDY** matching algorithm is well-suited for implementation in the *streaming model of computation*. A streaming algorithm processing a graph  $G = (V, E)$  with  $|V| = n$  receives a potentially adversarially ordered sequence of the edges of the input graph, and the objective

---

<sup>1</sup> Corresponding author



is to solve a graph problem using as little space as possible. Many graph problems require space  $\Omega(n \log n)$  to be solved in the streaming model [28], and streaming algorithms that use space  $O(n \text{ poly } \log n)$  are referred to as *semi-streaming algorithms*. Multi-pass streaming algorithms process the input stream multiple times. Observe that GREEDY constitutes a one-pass semi-streaming algorithm for Maximum Matching with approximation factor  $\frac{1}{2}$ .

The Maximum Matching problem is the most studied graph problem in the streaming model, and despite intense research efforts, the GREEDY algorithm is the best one-pass streaming algorithm known today, even if space  $O(n^{2-\delta})$  is allowed, for any  $\delta > 0$ . Performing multiple passes over the input allows improving the approximation factor. The main questions addressed in the literature are: (1) What can be achieved in  $p$  passes, for small  $p$  (e.g.  $p \in \{2, 3\}$ ), and (2) How many passes are required in order to obtain a  $(1 - \epsilon)$ -approximation, for any  $\epsilon > 0$ . See Table 1 for an overview of the currently best results.

■ **Table 1** State of the art semi-streaming algorithms for Maximum Matching.

# passes	Approximation	det/rand	Reference	See also
<b>Bipartite Graphs</b>				
1	$\frac{1}{2}$	deterministic	GREEDY, folklore	
2	$2 - \sqrt{2} \approx 0.5857$	randomized	Konrad [24]	[25, 14, 21]
3	0.6067	randomized	Konrad [24]	[14, 21]
$\frac{2}{3\epsilon}$	$\frac{2}{3} - \epsilon$	deterministic	Kale and Tirodkar [21]	[16]
$O(\frac{1}{\epsilon^2} \log \log \epsilon)$	$1 - \epsilon$	deterministic	Ahn and Guha [2]	[13]
<b>General Graphs</b>				
1	$\frac{1}{2}$	deterministic	GREEDY, folklore	
2	0.53125	deterministic	Kale and Tirodkar [21]	[25]
$\frac{1}{\epsilon} O(\frac{1}{\epsilon})$	$1 - \epsilon$	deterministic	Tirodkar [29]	[26]
$O(\frac{1}{\epsilon^4} \log n)$	$1 - \epsilon$	deterministic	Ahn and Guha [2]	

Only few lower bounds are known: We know that one-pass semi-streaming algorithms cannot have an approximation factor larger than  $1 - \frac{1}{e}$  [22] (see also [18]). The only multi-pass lower bound known addresses the exact version of Maximum Matching, showing that computing a maximum matching in  $p$  passes requires space  $n^{1+\Omega(1/p)}/p^{O(1)}$  [20]. No lower bound is known for multiple passes and approximations, and, for example, the existence of a 2-pass 0.999-approximation semi-streaming algorithm has not yet been ruled out.

The GREEDY algorithm is the key building block of all algorithms referenced in Table 1 (including those mentioned in the “See also” column). In many cases, the presented algorithms collect edges by *solely* executing GREEDY on specific subgraphs in each pass and output a large matching computed from the edges produced by GREEDY. In this paper, we are interested in the limitations of this approach: How large a matching can be computed if GREEDY is executed at most  $p$  times?

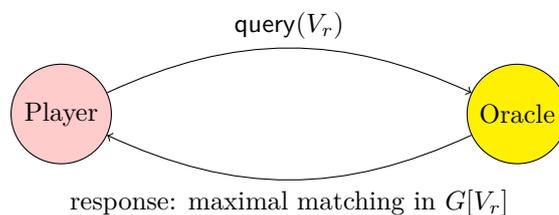
Known streaming algorithms apply GREEDY in different ways. For example, the 2-pass and 3-pass algorithms by Konrad [24] run GREEDY on randomly sampled subgraphs that depend on a previously computed maximal matching. The multi-pass algorithms by Ahn and Guha [2] maintain vertex weights  $\in [0, 1]$  over the course of the algorithm and run GREEDY on a *threshold subgraph*, i.e., on the set of edges  $uv$  so that the sum of the current weights associated with  $u$  and  $v$  is at most 1. The algorithm by Eggert et al. [13] runs GREEDY on an edge-induced subgraph in order to find augmenting paths.

In this paper, we initiate the study of lower bounds for restricted families of multi-pass streaming algorithms for Maximum Matching that are based on GREEDY. We start this line of research by addressing the probably simplest and most natural approach, which is

nevertheless surprisingly powerful: the class of deterministic algorithms that run GREEDY on a vertex-induced subgraph in each pass. Two known streaming algorithms fit our model:

1. A 3-pass 0.6-approximation streaming algorithm for bipartite graphs that is implicit in [16], explicitly mentioned in [25], and analyzed in [21]. Given a bipartite graph  $G = (A, B, E)$ , the algorithm first computes a maximal matching in  $G$ , i.e.,  $M \leftarrow \text{GREEDY}(G)$ . Then, the algorithm attempts to find length-3 augmenting paths by invoking GREEDY twice more:  $M_L \leftarrow \text{GREEDY}(G[A(M) \cup \overline{B(M)}])$ , where  $A(M)$  are the matched  $A$ -vertices and  $\overline{B(M)}$  are the unmatched  $B$ -vertices. Last,  $M_R \leftarrow \text{GREEDY}(\overline{A(M)}, B')$ , where  $B' \subseteq B(M)$  are those matched  $B$  vertices that are endpoints in length-2 paths in  $M_L \cup M$ . Kale and Tirodkar showed that  $M \cup M_L \cup M_R$  contains a 0.6-approximate matching [21]. We will denote this algorithm by 3ROUNDMATCH.
2. The  $(1 - \epsilon)$ -approximation  $O(\frac{1}{\epsilon^5})$ -passes streaming algorithm for bipartite graphs by Eggert et al. [13] can be adapted to fit our model using  $O(\frac{1}{\epsilon^5})$  invocations of GREEDY.

We abstract this approach as a game between a player and an oracle: Let  $G$  be a graph with vertex set  $V$ . The player initially knows  $V$ . In each round  $i$  the player sends a query  $\text{query}(V_i)$  to the oracle, where  $V_i \subseteq V$ . The oracle returns a maximal matching in the vertex-induced subgraph  $G[V_i]$ . For this model to yield lower bounds for the streaming model, we impose that the oracle is *streaming-consistent*, i.e., there exists a stream of edges  $\pi$  so that the oracle's answers to the queries  $(\text{query}(V_i))_i$  equal runs of GREEDY on the respective substream of edges  $G[V_i]$  of  $\pi$  (see preliminaries for a more detailed definition). We denote this model as the *vertex-query model* (as opposed to an edge-query model, where the player may ask for maximal matchings in a subgraph spanned by a subset of edges).



■ **Figure 1** Illustration of the game between the player and oracle in the vertex-query model.

**Our Results.** In bipartite graphs, we show that at least 3 rounds are required to improve on the approximation factor of  $1/2$ , and we give a lower bound of 0.6 on the approximation factor of 3 round algorithms. This is optimal, as demonstrated by the previously mentioned algorithm 3ROUNDMATCH. We also show that  $\Omega(\frac{1}{\epsilon})$  rounds are required for computing a  $(1 - \epsilon)$ -approximation. This polynomial lower bound is in line with the poly  $\frac{1}{\epsilon}$  rounds upper bound by Eggert et al. [13]. Last, we demonstrate that our query model is not well-suited to general graphs: We show that improving on a factor of  $1/2$  requires  $\Omega(n)$  rounds.

**Further Related Work.** Besides the adversarial one-pass and multi-pass streaming models, Maximum Matching has also been studied in the random order [25, 24, 17, 4, 15, 8] and the insertion-deletion settings [23, 9, 6, 12]. In the random order model, where edges arrive in uniform random order, Konrad et al. [25] were the first to give a semi-streaming algorithms with approximation ratio above  $1/2$ . Very recently, Bernstein showed that an approximation ratio of  $2/3$  can be achieved in random order streams [8]. In light of the lower bound

by Kapralov [22], this result separates the adversarial and the random order settings. In insertion-deletion streams, edges that have previously been inserted may be deleted again. Assadi et al. [6] showed that, up to sub-polynomial factors, space  $n^{2-3\epsilon}$  is necessary and sufficient for computing a  $n^\epsilon$ -approximation (see [12] for a slightly improved lower bound).

Many works allow only query access to the input graph. For example, cross-additive queries, bipartite independent set queries, additive queries, cut-queries, and edge-detection queries have been considered [19, 3, 11, 10, 7, 27, 1], however, mainly for graph reconstruction problems. Very recently, linear queries and or-queries have been considered for graph connectivity [5].

**Outline.** In Section 2, we give notation and definitions. We also define the vertex-query model and provide a construction mechanism that ensures that our oracles are streaming-consistent. Then, in Section 3 we prove that 3 rounds are required to improve on  $1/2$  and give a lower bound of  $0.6$  on the approximation ratio achievable in three rounds. In Section 4, we show that  $\Omega(\frac{1}{\epsilon})$  rounds are needed for computing a  $(1-\epsilon)$ -approximation, and in Section 5 we show that improving on  $\frac{1}{2}$  in general graphs requires  $\Omega(n)$  rounds. Finally, we conclude in Section 6 and give open questions.

## 2 Preliminaries

**Matchings.** Let  $G = (V, E)$  be a graph with  $|V| = n$ . A *matching*  $M \subseteq E$  is a subset of vertex-disjoint edges. Matching  $M$  is *maximal* if for every  $e \in E \setminus M : M \cup \{e\}$  is not a matching. A *maximum matching* is one of largest cardinality. If the size of a matching  $M$  is  $n/2$ , i.e., it matches all vertices of the graph, then  $M$  is a *perfect matching*.

**Notation.** We write  $V(M)$  to denote the set of vertices incident to the edges of a matching  $M$ . For a subset of vertices  $V' \subseteq V$ , we denote by  $G[V']$  the *vertex-induced subgraph* of  $G$  by vertices  $V'$ , i.e.,  $G[V'] = (V', (V' \times V') \cap E)$ . For a set of edges  $E' \subseteq E$ , we denote by  $OPT(E')$  the size of a maximum matching in the subgraph of  $G$  spanned by the edges  $E'$ . For an integer  $n$ , we define  $[n] := \{1, 2, \dots, n\}$ .

**The Vertex-query Model.** In the *vertex-query model*, a player and an oracle play a rounds-based matching game on a vertex set  $V$  of size  $n$  that is initially known to both parties. Over the course of the game, the oracle makes up a graph  $G = (V, E)$ . The objective of the player is to learn a large matching in  $G$ . The way the player learns edges is as follows:

In each round  $1 \leq i \leq r$ , where  $r$  is the total number of rounds played, the player submits a query  $\text{query}(V_i)$  to the oracle, for some  $V_i \subseteq V$ . The oracle then determines a set of edges  $M_i$ , which is guaranteed to be a maximal matching in the vertex-induced subgraph  $G[V_i]$ . Observe that in doing so, the oracle not only commits to the fact that  $M_i \subseteq E$ , but also that the vertices  $V_i \setminus V(M_i)$  form an independent set (which follows from the fact that  $M_i$  is maximal). Furthermore, we impose that the answers to all queries are consistent with graph  $G$  and that  $G$  has a perfect matching.

After the  $r$  query rounds, the player reports a largest matching  $M_P$  that can be formed using the edges  $\cup_{i \leq r} M_i$ . The *approximation ratio* of the solution obtained is  $|M_P| / (\frac{1}{2}n)$ .

We are interested in oracles that are consistent with the streaming model. We say that an oracle is *streaming-consistent*, if there exists an ordering  $\pi$  of the edges  $E$  so that, for every round  $i$ ,  $M_i$  is produced by running GREEDY on the substream of  $\pi$  consisting of the edges of  $G[V_i]$ . We will ensure that all our oracles are streaming-consistent.

**Construction of Streaming-consistent Oracles.** We will construct streaming-consistent oracles as follows. Upon query  $V_1$ , the oracle answers with  $M_1$  and places  $M_1$  in the beginning of the stream  $\pi$ . Next, given query  $V_i$ , for some  $i \geq 2$ , the oracle first runs GREEDY on the substream of  $\pi$  consisting of the edges  $G[V_i]$  which produces an intermediate matching  $M'$ , thereby attempting to match  $V_i$  using edges of previous matchings  $\cup_{j < i} M_j$ . The oracle then extends  $M'$  to a matching  $M_i$ . Edges  $M_i \setminus M'$  are then introduced at the end of the stream  $\pi$ . This construction procedure guarantees that our oracles are streaming-consistent. Furthermore, it allows us to simplify our arguments, since it is enough to restrict our considerations to queries with the following property:

► **Observation 1.** *Suppose that the oracle is constructed as above. Then, given the sequence of queries  $V_1, \dots, V_r$  and matchings  $M_1, \dots, M_r$ , there exists a sequence of queries  $\tilde{V}_1, \dots, \tilde{V}_r$  that produces matchings  $\tilde{M}_1, \dots, \tilde{M}_r$  such that:*

- *The player learns the same set of edges, i.e., for every  $i \leq r : \cup_{j \leq i} M_j = \cup_{j \leq i} \tilde{M}_j$ , and*
- *No query  $\tilde{V}_i$  contains a pair of vertices  $u, v$  such that  $uv \in \cup_{j < i} \tilde{M}_j$ .*

We can therefore assume that the player never includes a pair of vertices  $u, v$  into a query so that the edge  $uv$  is contained in a previous answer from the oracle.

### 3 Lower Bound for Few Round Algorithms in Bipartite Graphs

In this section, we show that the player cannot produce an approximation ratio better than  $\frac{1}{2}$  in two rounds, even on bipartite graphs. We also show that three rounds do not allow for an approximation ratio better than 0.6, which is achieved by the algorithm 3ROUNDMATCHING.

In order to keep track of the information learned by the player, we will make use of *structure graphs*, which we discuss first.

#### 3.1 Structure Graphs

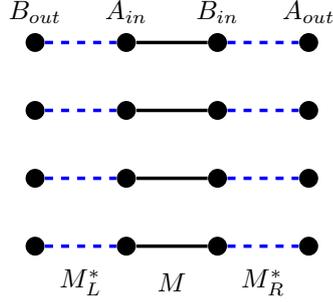
Observe that when the oracle answers the query  $\text{query}(V_i)$  and returns a maximal matching  $M_i$ , the player not only learns that the edges  $M_i$  are contained in the input graph  $G$ , but also learns that the vertices  $V_i \setminus V(M_i)$  form an independent set in  $G$  (due to the maximality of  $M_i$ ). We maintain the structure learned by the player and the structure committed to by the oracle (which do not have to be identical) using *structure graphs*:

► **Definition 2** (Structure graph). *A 4-tuple  $(A, B, E, F)$  is a **bipartite structure graph** if:*

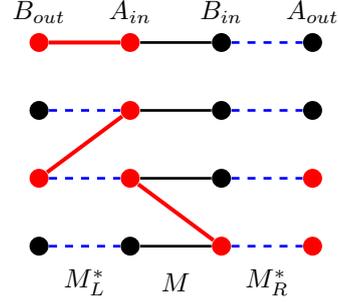
- *$A, B$  are disjoint sets of vertices,*
- *$E, F$  are disjoint sets of edges such that  $(A, B, E)$  and  $(A, B, F)$  are bipartite graphs,*
- *The structure graph admits a perfect matching, i.e., there exists a set of edges  $M^*$  such that  $M^* \cap F = \emptyset$  and  $M^*$  is a perfect matching in the bipartite graph  $(A, B, E \cup M^*)$ .*

From the perspective of the player, the set  $E$  corresponds to the edges returned by the oracle so far, i.e.,  $E = \cup_{j \leq i} M_j$ , and the set  $F$  corresponds to guaranteed *non-edges*, i.e.,  $F = \cup_{j \leq i} C(V_i \setminus V(M_j))$ , where  $C(V')$  denotes a biclique (respecting the bipartition  $A, B$ ) among the vertices  $V'$ .

In the following, we will denote the structure graph after round  $i$  learned by the player by  $\tilde{H}_i = (A, B, \tilde{E}_i, \tilde{F}_i)$ , i.e.,  $\tilde{E}_i = \cup_{j \leq i} M_j$  and  $\tilde{F}_i = \cup_{j \leq i} C(V_i \setminus V(M_j))$ . The oracle will also maintain a sequence of structure graphs  $(H_i)_i$  with  $H_i = (A, B, E_i, F_i)$  such that  $H_i$  *dominates*  $\tilde{H}_i$ , for every  $1 \leq i \leq r$ . We say that a structure graph  $H = (A, B, E, F)$  dominates a structure graph  $\tilde{H} = (A, B, \tilde{E}, \tilde{F})$ , if  $\tilde{E} \subseteq E$  and  $\tilde{F} \subseteq F$ . This notion allows the oracle to commit to edges and non-edges that the player has not yet learned. This domination property allows us to simplify our arguments.



■ **Figure 2** Illustration of the structure graph  $H_1$  on a graph on 16 vertices. The matching  $M$  is half the size of the matching  $M^* = M_L^* \cup M_R^*$ .



■ **Figure 3** Matching  $M_2$  (in red) returned by the oracle. The red vertices constitute  $A_2 \cup B_2$ , i.e., the vertices of the second query. The case  $|A_2^{in}| \geq |B_2^{in}|$  is illustrated here. We see that no edges from  $B_{in} \times A_{out}$  are returned, and that  $M_2$  does not allow us to increase the size of  $M$ .

In our lower bound arguments, we make use of the following two assumptions:

► **Assumption 1.** *After round  $i$ , the player knows the structure graph  $H_i$ .*

This is a valid assumption since  $H_i$  dominates  $\tilde{H}_i$  and thus contains at least as much information as  $\tilde{H}_i$ . This assumption therefore only strengthens the player. Furthermore, we will also assume a slightly strengthened property of the property discussed in Observation 1:

► **Assumption 2.** *For every  $1 \leq i \leq r$ , we assume that query  $V_i$  does not contain a pair of vertices  $u, v \in V_i$  such that  $uv \in E_{i-1}$ .*

This is a valid assumption, since if such a pair  $u, v$  of vertices existed in  $V_i$ , the oracle could simply match  $u$  to  $v$  in  $M_i$  and the algorithm would not learn any new information.

Last, observe that the approximation ratio of the player's strategy is completely determined by  $H_r$ , the oracle's structure graph after the last round. Since  $H_r$  dominates  $\tilde{H}_r$ , the player's largest matching is of size at most  $OPT(E_r)$ . Since by definition of a structure graph,  $H_r$  admits a perfect matching, the approximation ratio achieved is  $2 \cdot OPT(E_r)/n$ .

### 3.2 Lower Bound for Two Rounds

Assume that  $n$  is a multiple of 4. The player and the oracle play the matching game on a bipartite vertex set  $V = A \dot{\cup} B$  with  $|A| = |B| = n/2$ . Consider the structure graph:

$$H_1 = (A_{in} \cup A_{out}, B_{in} \cup B_{out}, M, A_{out} \times B_{out}),$$

where  $|A_{in}| = |A_{out}| = |B_{in}| = |B_{out}| = n/4$ , and  $M$  is a perfect matching between  $A_{in}$  and  $B_{in}$ . Observe that there exists an  $M^*$  outside  $A_{out} \times B_{out}$  such that  $M^*$  is a perfect matching in  $(A, B, M \cup M^*)$ , namely,  $M^*$  consists of the two perfect matchings  $M_L^*$  connecting  $B_{out}$  to  $A_{in}$  and  $M_R^*$  connecting  $B_{in}$  to  $A_{out}$ . See Figure 2 for an illustration.

We have:

► **Lemma 3.** *There is a structure graph isomorphic to  $H_1$  that dominates  $\tilde{H}_1$ .*

**Proof.** Denote the first query by  $A_1, B_1$  ( $A_1 \subseteq A$ , and  $B_1 \subseteq B$ ). We will argue that we can relabel the sets  $A_{in}, A_{out}, B_{in}, B_{out}$  so that  $H_1$  dominates  $\tilde{H}_1$ :

If  $A_1 \leq n/4$  then let  $A_{in}$  be an arbitrary subset of the  $A$  vertices of size  $n/4$  that contains  $A_1$ , and let  $A_{out}$  be the remaining  $A$ -vertices. If  $A_1 > n/4$  then let  $A_{out}$  be an arbitrary subset of  $A$  vertices of size  $n/4$  that contains  $A \setminus A_1$ , and let  $A_{in}$  be the remaining  $A$ -vertices. Proceed similarly for  $B_1$ . The oracle returns the subset  $M_1 \subseteq M$  where each edge has one endpoint in  $A_1$  and one endpoint in  $B_1$ , which is clearly maximal given that edges in  $A_{out} \times B_{out}$  are forbidden. ◀

Since  $OPT(M) = |M| = \frac{1}{4}n$ , Lemma 3 implies the unsurprising fact that no one round algorithm has an approximation ratio better than  $\frac{2 \cdot \frac{1}{4}n}{n} = \frac{1}{2}$ . We argue now that an additional round does not help with increasing the approximation factor.

► **Theorem 4.** *The best approximation ratio achievable in two rounds is  $1/2$ .*

**Proof.** Let  $A_2, B_2$  be the vertices of the second query. By Lemma 3,  $H_1$  dominates  $\tilde{H}_1$ , and by Assumption 1 we can assume that the player already knows  $H_1$ . Let  $A_2^{in} = A_2 \cap A_{in}$ ,  $A_2^{out} = A_2 \cap A_{out}$  and define  $B_2^{in}$  and  $B_2^{out}$  similarly.

Suppose first that  $|A_2^{in}| \geq |B_2^{in}|$ . Then the oracle returns a matching  $M_2$  that matches an arbitrary subset of  $A_2^{in}$  of size  $|B_2^{in}|$  to  $B_2^{in}$ , and matches  $\max\{|B_2^{out}|, |A_2^{in}| - |B_2^{in}|\}$  of the remaining  $A_2^{in}$  vertices arbitrarily to vertices in  $B_2^{out}$ . In doing so, either all  $A_2^{in}$  vertices or all  $B_2$  vertices are matched. Since  $H_1$  indicates that there are no edges connecting the “out”-vertices,  $M_2$  is therefore maximal.

Observe further that  $M \cup M_2$  does not match any vertex in  $A_{out}$ , and, hence, only half of the  $A$ -vertices are matched in  $M \cup M_2$ . The player thus cannot report any matching of size larger than  $|M|$ , which constitutes a  $1/2$ -approximation.

Last, the case  $|A_2^{in}| < |B_2^{in}|$  is identical with roles of  $A$  and  $B$  vertices reversed. ◀

### 3.3 Lower Bound for Three Rounds

In this section, we work with a vertex set  $V = A \dot{\cup} B$  with  $|A| = |B| = 5$  (and thus  $|V| = n = 10$ ). By choosing disjoint copies of this vertex set, our result can be extended to graphs with an arbitrarily large number of vertices.

**First Query.** Similar to the two round case, we define the structure graph  $H_1 = (A_{in} \cup A_{out}, B_{in} \cup B_{out}, M, A_{out} \times B_{out})$ , however, this time  $|A_{in}| = |B_{in}| = 3$  and  $|A_{out}| = |B_{out}| = 2$ . The matching  $M$  matches  $A_{in}$  to  $B_{in}$ , see Figure 4a.

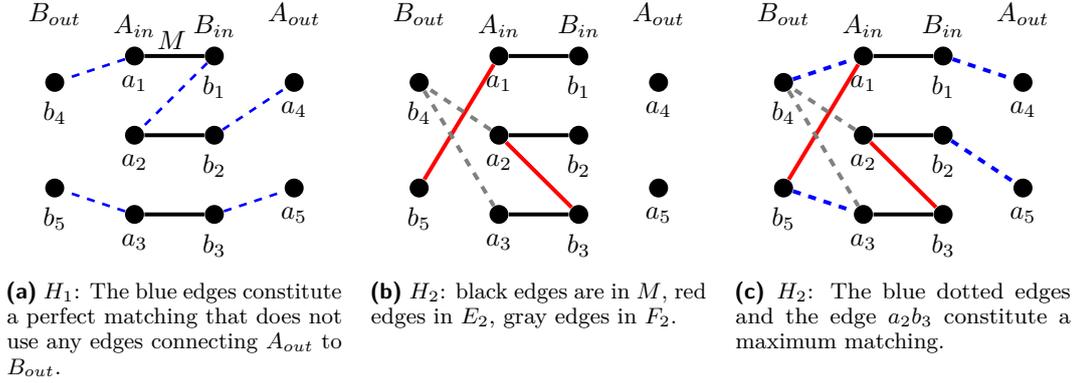
It shall be convenient to assign labels to the vertices in our structure graph. In our arguments below, in order to avoid symmetric cases, we relabel the vertices of our structure graph as we see fit, however, we always ensure that the structure graph after relabeling is isomorphic to the structure graph before the relabeling.

First, similar to Lemma 3, it is not hard to see that a structure graph isomorphic to  $H_1$  dominates  $\tilde{H}_1$  (proof omitted).

► **Lemma 5.** *There is a structure graph isomorphic to  $H_1$  that dominates  $\tilde{H}_1$ .*

**Second Query.** We assume that the player knows  $H_1$  after the first query (Assumption 1). Next, we define structure graph  $H_2 = (A_{in} \cup A_{out}, B_{in} \cup B_{out}, M \cup E_2, A_{out} \times B_{out} \cup F_2)$ , where  $E_2 = \{a_1b_5, a_2b_3\}$ , and  $F_2 = \{a_2b_4, a_3b_4\}$ . It is easy to see that  $H_2$  is indeed a structure graph (see Figures 4b and 4c).

We shall prove that there is a structure graph isomorphic to  $H_2$  that dominates  $\tilde{H}_2$ . Lemma 6 considers the case when the second query  $V_2$  contains exactly three “in”-vertices, i.e., vertices from  $A_{in} \cup B_{in}$ , and Lemma 7 considers the case when there are fewer “in”-vertices.



■ **Figure 4** Illustrations of structure graphs  $H_1$  and  $H_2$ .

By Assumption 2, we do not need to consider the cases when more than three “in”-vertices are contained in  $V_2$  since then  $V_2$  necessarily contains a pair of vertices  $u, v$  such that  $uv \in M$ .

► **Lemma 6.** *If the player queries exactly 3 “in”-vertices (i.e., vertices from  $A_{in} \cup B_{in}$ ) in their second query then there exists a structure graph isomorphic to  $H_2$  that dominates  $\tilde{H}_2$ .*

**Proof.** The player can either query more vertices in  $A_{in}$  or in  $B_{in}$ , and these cases are symmetrical. Hence we only consider the case when the player queries more vertices in  $A_{in}$ . Due to Assumption 2, for queries that contain vertices in both  $A_{in}$  and  $B_{in}$ , we assume these vertices do not form any edges seen in  $M$ .

Since we will not match any vertices in  $A_{out}$ , we do not need to distinguish between cases where the player queries different numbers of vertices in  $A_{out}$ . We distinguish between the following cases:

1. Player queries all vertices in  $A_{in}$  and the query includes  $b_5$ : the oracle returns  $M_2 = \{a_1b_5\}$ .
2. Player queries all vertices in  $A_{in}$  and only  $b_4$  in  $B_{out}$ : relabel  $b_4$  as  $b_5$  and proceed as in case (1).
3. Player queries all vertices in  $A_{in}$  and no vertices in  $B_{out}$ : the oracle returns  $M_2 = \emptyset$ .
4. Player queries two vertices in  $A_{in}$ , one vertex in  $B_{in}$  and the query includes  $b_5$ : relabel the “in” vertices so that after relabeling the vertices  $a_1, a_2$  and  $b_3$  are included in the query. The oracle returns  $M_2 = E_2$ .
5. Player queries two vertices in  $A_{in}$ , one vertex in  $B_{in}$  and only  $b_4$  in  $B_{out}$ : relabel  $b_4$  as  $b_5$  and proceed as in case (4).
6. Player queries two vertices in  $A_{in}$ , one vertex in  $B_{in}$  and no vertices in  $B_{out}$ : relabel “in” vertices so that after relabeling the vertices  $a_2$  and  $b_3$  are included in the query. The oracle returns  $M_2 = \{a_2b_3\}$ .

In all cases considered, observe that  $M_2 \subseteq E_2$ . Further, edges  $F_2$  ensure that  $M_2$  is maximal. ◀

We argue now that querying three “in”-vertices in the second round is best possible in the sense that querying fewer (or more) “in”-vertices does not yield more information.

► **Lemma 7.** *If the player queries fewer than 3 “in”-vertices (i.e., vertices from  $A_{in} \cup B_{in}$ ) then there exists a structure graph isomorphic to  $H_2$  that dominates  $\tilde{H}_2$ .*

**Proof.** Clearly if the player does not query any “in”-vertices, no matching will be found i.e.  $M_2 = \emptyset$ . If the player queries exactly one vertex in  $A_{in}$ , we can relabel this vertex as  $a_1$  and if the query contains a vertex in  $B_{out}$ , relabel this one to be  $b_5$ . Then the matching

found will be a subset of  $E_2$ . If the player queries exactly two “in” vertices there are two cases to consider. If they are both in  $A_{in}$ , we ensure one of these vertices is  $a_1$  by relabeling, and, if at least one vertex in  $B_{out}$  is queried, potentially relabel this vertex to be  $b_5$  and return the edge  $a_1b_5$ . If the player queried one vertex in  $A_{in}$  and one in  $B_{in}$ , we relabel these vertices as  $a_2, b_3$  and return the edge between them,  $a_2b_3$ . Hence the edges learned by the player are always a subset of  $E_2$ . In all cases considered, edges  $F_2$  ensure that matching  $M_2$  is maximal. ◀

**Third Query.** We assume that the player knows structure graph  $H_2$ . Similar to the second query, we distinguish between the cases where the player queries exactly three “in”-vertices and fewer “in”-vertices. Again, by Assumption 2, we do not need to consider the case where the player queries more than three “in”-vertices. In the following proofs, we will define different structure graphs  $H_3$  that depend on the individual query.

► **Lemma 8.** *If the player queries exactly 3 “in”-vertices in the third round, then the player cannot output a matching of size larger than 3.*

**Proof.** We provide the oracle’s answers when the player queries exactly three “in”-vertices. Among those cases, there are three cases to consider where the player queries more vertices in  $B_{in}$  than in  $A_{in}$ :

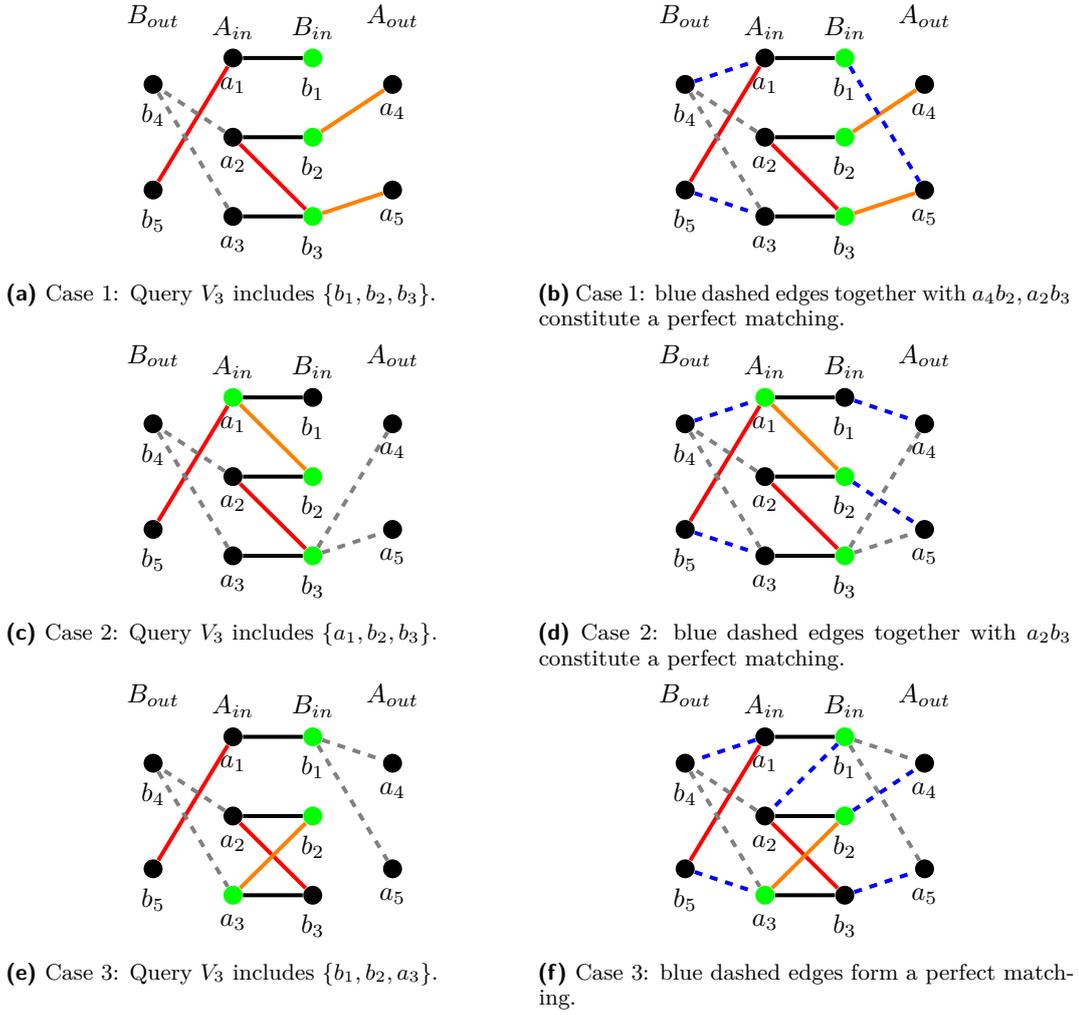
1. **Case 1:** Player queries  $b_1, b_2, b_3$ . The oracle defines  $H_3 = (A, B, E_3, F_3)$  such that  $E_3 = M \cup E_2 \cup \{a_4b_2, a_5b_3\}$  and  $F_3 = A_{out} \times B_{out} \cup F_2$ . If the player queried both vertices in  $A_{out}$ , the oracle returns  $M_3 = \{a_4b_2, a_5b_3\}$ . Otherwise  $M_3$  would consist of one or zero edges depending on the player’s query. In particular, we have  $M_3 \subset E_3$ .  
In cases 2 and 3, we do not define any edges involving vertices from  $A_{out}$  or  $B_{out}$ , so the oracle proceeds regardless of which vertices in  $A_{out}, B_{out}$  the player queried.
2. **Case 2:** Player queries  $a_1, b_2, b_3$ . The oracle defines  $H_3 = (A, B, E_3, F_3)$  such that  $E_3 = M \cup E_2 \cup \{a_1b_2\}$  and  $F_3 = A_{out} \times B_{out} \cup F_2 \cup \{a_4b_3, a_5b_3\}$ . The oracle returns  $M_3 = \{a_1b_2\}$ .
3. **Case 3:** Player queries  $b_1, b_2, a_3$ . The oracle defines  $H_3 = (A, B, E_3, F_3)$  such that  $E_3 = M \cup E_2 \cup \{a_3b_2\}$  and  $F_3 = A_{out} \times B_{out} \cup F_2 \cup \{a_4b_1, a_5b_1\}$ . The oracle returns  $M_3 = \{a_3b_2\}$ .

Observe that the case  $b_1, a_2, b_3 \in V_3$  is not relevant, since  $a_2b_3 \in M_2$  and Assumption 2. Figure 5 shows that in these three cases,  $H_3$  is a structure graph and the largest matching that the player thus able to return is of size 3.

If the player queries more vertices in  $A_{in}$  than in  $B_{in}$ , we will argue that the player will not learn any edges connecting to vertices in  $A_{out}$ , and since the player then only holds edges incident to 3 of the 5  $A$ -vertices, the player cannot report a matching larger than of size 3.

If the player queries all three vertices in  $A_{in}$  then he clearly cannot learn any edges connecting to  $A_{out}$ . If the player queries a vertex in  $B_{in}$ , note that we can match it with a vertex queried in  $A_{in}$ , and there will be no vertices left to match with vertices in  $A_{out}$  (see Figure 6). Since no more non-edges are defined, it is easy to see that edges can be added to create a perfect matching. ◀

► **Lemma 9.** *If the player queries fewer than three “in”-vertices in the third round, then the player cannot output a matching of size larger than 3.*

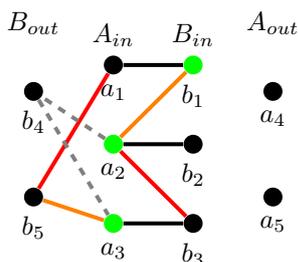


■ **Figure 5** Round 3 cases. Green vertices are queried by the player in round 3. Red edges are in  $E_2 \setminus E_1$ , orange is  $E_3 \setminus E_2$ , grey is  $F_3$ . The blue dashed edges can be added to the graph to create a perfect matching.

**Proof.** We distinguish the following cases:

1. If the player queries no “in” vertices, this is obvious, and we would have  $M_3 = \emptyset$ .
2. If the player queries exactly one “in” vertex, the only possible way to obtain a larger matching than one of size 3 is to find an edge incident to  $b_1$ , i.e., by querying  $b_1$ , but we can define  $F_3 = A_{out} \times B_{out} \cup F_2 \cup \{a_4b_1, a_5b_1\}$  and then  $M_3 = \emptyset$ .
3. If the player queries one vertex in  $A_{in}$  and one in  $B_{in}$ , we can connect them by an edge, say  $e$ , and then  $M_3 = \{e\}$  does not help increasing the size of a matching.
4. If the player queries two vertices in  $A_{in}$ , the player will not be able to learn any edges to vertices in  $A_{out}$ , and so  $A_{out}$  remains unmatched, which implies that the player cannot return a matching of size larger than 3.
5. If the player queries two vertices in  $B_{in}$ , the oracle defines  $H_3$  as in Case 1 of Lemma 8, and the matching returned is a subset of  $E_3$ . ◀

Hence we have shown that no matter what queries are made in the second and third rounds, the player cannot increase the size of the matching learned within the 10-vertex subgraph. This then holds for a graph with  $|A| = |B| = n$  where  $5|n$  and the theorem follows.



■ **Figure 6** An example of how the oracle behaves when the player queries more vertices in  $A_{in}$  than in  $B_{in}$  during the third round. Green vertices are queried by the player. Red edges are in  $E_2 \setminus E_1$ , orange is  $E_3 \setminus E_2$ , gray is  $F_3$ . The player learns no edges incident to  $A_{out}$  and can therefore only report a matching of size 3.

► **Theorem 10.** *The best approximation factor achievable in three rounds is  $3/5$ .*

**4**  $(1 - \epsilon)$ -approximation in Bipartite Graphs Requires  $\Omega(\frac{1}{\epsilon})$  Rounds

Let  $G_c = (A, B, E)$  with  $A = B = [c]$  be the *semi-complete* graph on  $2c$  vertices, i.e., vertices  $a \in A$  and  $b \in B$  are connected if and only if  $b \geq a$ . Observe that  $G_c$  has a unique perfect matching  $M^* = \{(i, i) \in E \mid i \in [c]\}$ .

Let  $G$  be the disjoint union of  $n/(2c)$  copies of  $G_c$  (assuming for simplicity that  $n$  is a multiple of  $2c$ ). We will refer to a copy of  $G_c$  in  $G$  as a gadget. We now show that computing a  $(1 - \epsilon)$ -approximation requires  $\Omega(\frac{1}{\epsilon})$  queries on  $G$ .

► **Theorem 11.** *Any query algorithm with approximation factor  $1 - \epsilon$  requires at least  $\frac{1}{\epsilon} - 1$  queries, even in bipartite graphs.*

**Proof.** Let  $c = \frac{1}{\epsilon} - 1$ . We consider the graph  $G$ . First, suppose that the algorithm does not compute a perfect matching in any of the  $n/(2c)$  gadgets. Then, the computed matching is of size at most  $\frac{c-1}{c} \frac{n}{2}$  and thus constitutes at best a  $\frac{c-1}{c} = 1 - \frac{\epsilon}{1-\epsilon} < 1 - \epsilon$  approximation. The algorithm therefore needs to compute a perfect matching in at least one gadget. Since all gadgets are disjoint, we now argue that it requires at least  $c$  queries in order to compute a perfect matching in one gadget. Consider thus the gadget  $G_c$  and denote by  $M^*$  the perfect matching in  $G_c$ . We claim that each query may produce at most one edge of the perfect matching  $M^*$  in  $G_c$ :

Indeed, let  $A' = \{a_1, a_2, \dots, a_k\} \subseteq A$  and  $B' = \{b_1, b_2, \dots, b_\ell\} \subseteq B$  be so that  $A' \cup B'$  is any query submitted to the oracle. Further, suppose that  $a_1 < a_2 < \dots < a_k$  and  $b_1 < b_2 < \dots < b_\ell$ . The oracle will return the following matching  $M$ :

$$M = \{a_i b_{\ell+1-i} \mid i \in [\min\{k, \ell\}]\} \cap E.$$

We will now argue that  $M$  is maximal and  $|M \cap M^*| \leq 1$ . To this end, let  $j$  be the largest index such that  $a_j b_{\ell+1-j} \in E$ , which is equivalent to  $j$  being the largest index so that  $a_j \leq b_{\ell+1-j}$ . Observe that since the  $(a_i)_i$  and  $(b_i)_i$  are increasing, we have  $a_{j'} b_{\ell+1-j'} \in E \Leftrightarrow j' \leq j$ , which also implies that vertices  $a_{j'}$  are matched, for every  $j' \leq j$ . Consider now a vertex  $a_q$ , for some  $q > j$ . Since  $a_{j+1} > b_{\ell-j}$  and  $a_q \geq a_{j+1}$ , it follows that there is no edge between  $a_q$  and any of the unmatched  $B'$ -vertices  $\{b_1, b_2, \dots, b_{\ell-j}\}$ . This implies that the matching  $M$  is maximal. Next, suppose that  $M$  contains at least one edge from  $M^*$  and let  $q$  be the smallest index such that  $a_q = b_{\ell+1-q}$ , i.e.,  $(a_q, b_{\ell+1-q}) \in M^*$ . Then, for any  $q' > q$ , we have

$$a_{q'} > a_q = b_{\ell+1-q} > b_{\ell+1-q'},$$

## 26:12 Constructing Large Matchings via Query Access to a Maximal Matching Oracle

which implies that  $a_{q'} \neq b_{\ell+1-q'}$ . Hence, at most one edge from  $M^*$  is returned per query.

Last, we argue that the oracle can be made streaming-consistent: Consider any ordering of the edges so that edge  $ij$  arrives before edge  $ik$ , for every  $k < j$ . ◀

Using the oracle described in the previous proof on a single gadget  $G_{n/2}$ , we obtain the following corollary:

► **Corollary 12.** *Any query algorithm that produces a maximum matching requires at least  $n/2$  queries (on a graph on  $n$  vertices), even on bipartite graphs.*

### 5 Improving on $1/2$ in General Graphs Requires $\Omega(n)$ Queries

Let  $G$  be a *bomb graph* on  $n$  ( $n$  even) vertices  $U \cup V$  with  $|U| = |V| = [n/2]$ , where  $G[V]$  is a clique,  $G[U]$  is an independent set, and  $u \in U$  and  $v \in V$  are connected if and only if  $u = v$  ( $U$  and  $V$  are connected via a perfect matching). Denote by  $M^*$  the perfect matching between  $U$  and  $V$  and by  $C$  the edges of the clique  $G[V]$ .

In the next lemma, we show that any large matching in  $G$  must contain a large number of edges from  $M^*$ .

► **Lemma 13.** *Let  $M$  be a matching in  $G$ . Then:  $|M| \leq \frac{n}{4} + \frac{1}{2}|M \cap M^*|$ .*

**Proof.** Observe that  $|M| = |M \cap M^*| + |M \cap C|$ , and since there are  $n/2 - |M \cap M^*|$  vertices in  $V$  that are not matched to a vertex in  $U$ , we have  $|M \cap C| \leq (n/2 - |M \cap M^*|)/2$ . Hence:

$$|M| = |M \cap M^*| + |M \cap C| \leq |M \cap M^*| + (n/2 - |M \cap M^*|)/2 = \frac{n}{4} + \frac{1}{2}|M \cap M^*|. \quad \blacktriangleleft$$

► **Theorem 14.** *Any  $r$ -round query algorithm on general graphs has approximation ratio at most  $\frac{1}{2} + \frac{r}{n}$  (on an  $n$ -vertex input graph).*

**Proof.** Consider an arbitrary query  $U' \cup V'$  so that  $U' \subseteq U$  and  $V' \subseteq V$ . The oracle returns the following matching: First, the oracle arbitrarily pairs up all vertices of  $V'$  except possibly one in case  $|V'|$  is odd. Let  $M$  denote this matching. If  $|V'|$  is even then  $M$  is returned. Suppose now that  $|V'|$  is odd and let  $v \in V'$  be the vertex that is not matched in  $M$ . Then, if  $v$ 's partner  $u \in U$  in  $M^*$  is contained in  $U'$ , then return  $M \cup \{uv\}$ , otherwise return  $M$ .

It is easy to see that, by construction, the returned matching is maximal and contains at most one edge from  $M^*$ . Hence, in  $r$ -rounds the algorithm can learn at most  $r$  edges from  $M^*$ . By Lemma 13, the returned matching is therefore of size at most  $\frac{n}{4} + \frac{1}{2}r$ , which constitutes a  $\frac{1}{2} + \frac{r}{n}$ -approximation.

The oracle can be made streaming-consistent: Consider any edge order where we first have edges  $C$  in arbitrary order followed by  $M^*$  in arbitrary order. ◀

### 6 Conclusion

In this paper, we introduced a new query model that allows us to prove lower bounds for streaming algorithms for **Maximum Matching** that repeatedly run the **GREEDY** matching algorithm on a vertex-induced subgraph of the input graph. We showed that the three rounds algorithm **3ROUNDMATCH** with approximation factor 0.6 is optimal for this class of algorithms. We also showed that computing a  $(1 - \epsilon)$ -approximation in bipartite graphs requires  $\Omega(\frac{1}{\epsilon})$  rounds, and computing an approximation strictly better than  $\frac{1}{2}$  in general graphs requires  $\Omega(n)$  rounds. We conclude with open questions:

- Can we prove that computing a maximum matching in the vertex-query model in bipartite graphs requires  $\Omega(n^2)$  rounds, or is there an algorithm that requires only  $o(n^2)$  rounds?
- Can we prove a  $\Omega(\frac{1}{\epsilon^2})$  lower bound for computing a  $(1 - \epsilon)$ -approximation in bipartite graphs?

---

## References

- 1 Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. In Aurélien Garivier and Satyen Kale, editors, *Algorithmic Learning Theory, ALT 2019, 22-24 March 2019, Chicago, Illinois, USA*, volume 98 of *Proceedings of Machine Learning Research*, pages 3–30. PMLR, 2019. URL: <http://proceedings.mlr.press/v98/abasi19a.html>.
- 2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011). doi:10.1016/j.ic.2012.10.006.
- 3 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004. doi:10.1137/S0097539702420139.
- 4 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.
- 5 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and or queries, 2020. arXiv:2007.06098.
- 6 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.ch93.
- 7 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge Estimation with Independent Set Oracles. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:21, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.38.
- 8 Aaron Bernstein. Improved Bounds for Matching in Random-Order Streams. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.12.
- 9 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344. SIAM, 2016. doi:10.1137/1.9781611974331.ch92.
- 10 Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 797–818. JMLR.org, 2013. URL: <http://proceedings.mlr.press/v30/Choi13.html>.
- 11 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC*

- '08, page 749–758, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374484.
- 12 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In *35th Computational Complexity Conference, CCC 2020*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
  - 13 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1–2):490–508, June 2012.
  - 14 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In Carlotta Domeniconi, Francesco Gullo, Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain*, pages 608–614. IEEE Computer Society, 2016. doi:10.1109/ICDMW.2016.0092.
  - 15 Alireza Farhadi, MohammadTaghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '20*, page 1773–1785, USA, 2020. Society for Industrial and Applied Mathematics.
  - 16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005. doi:10.1016/j.tcs.2005.09.013.
  - 17 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19*, page 491–500, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331603.
  - 18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.
  - 19 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi:10.1007/s004530010033.
  - 20 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. doi:10.1007/s00453-016-0138-7.
  - 21 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.
  - 22 Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.
  - 23 Christian Konrad. Maximum matching in turnstile streams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015. doi:10.1007/978-3-662-48350-3\_70.
  - 24 Christian Konrad. A Simple Augmentation Method for Matchings with Applications to Streaming Algorithms. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 74:1–74:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2018.74.

- 25 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0\_20.
- 26 Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th International Conference on Randomization and Computation: Algorithms and Techniques, APPROX'05/RANDOM'05*, page 170–181, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11538462\_15.
- 27 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2018.39.
- 28 Xiaoming Sun and David P. Woodruff. Tight Bounds for Graph Problems in Insertion Streams. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 435–448, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.435.
- 29 Sumedh Tirodkar. Deterministic Algorithms for Maximum Matching on General Graphs in the Semi-Streaming Model. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.39.



# Planted Models for the Densest $k$ -Subgraph Problem

**Yash Khanna**

Indian Institute of Science, Bangalore, India  
yashkhanna@iisc.ac.in

**Anand Louis**

Indian Institute of Science, Bangalore, India  
anandl@iisc.ac.in

---

## Abstract

Given an undirected graph  $G$ , the DENSEST  $k$ -SUBGRAPH problem (DkS) asks to compute a set  $S \subset V$  of cardinality  $|S| \leq k$  such that the weight of edges inside  $S$  is maximized. This is a fundamental NP-hard problem whose approximability, inspite of many decades of research, is yet to be settled. The current best known approximation algorithm due to Bhaskara et al. (2010) computes a  $\mathcal{O}(n^{1/4+\epsilon})$  approximation in time  $n^{\mathcal{O}(1/\epsilon)}$ , for any  $\epsilon > 0$ .

We ask what are some “easier” instances of this problem? We propose some natural semi-random models of instances with a planted dense subgraph, and study approximation algorithms for computing the densest subgraph in them. These models are inspired by the semi-random models of instances studied for various other graph problems such as the independent set problem, graph partitioning problems etc. For a large range of parameters of these models, we get significantly better approximation factors for the DENSEST  $k$ -SUBGRAPH problem. Moreover, our algorithm recovers a large part of the planted solution.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Semidefinite programming; Theory of computation  $\rightarrow$  Discrete optimization; Theory of computation  $\rightarrow$  Graph algorithms analysis

**Keywords and phrases** Densest  $k$ -Subgraph, Semi-Random models, Planted Models, Semidefinite Programming, Approximation Algorithms, Beyond Worst Case Analysis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.27

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2004.13978>.

**Funding** *Anand Louis*: AL was supported in part by SERB Award ECR/2017/003296 and a Pratiksha Trust Young Investigator Award.

**Acknowledgements** We thank Rakesh Venkat for helpful discussions. We also thank the anonymous reviewers for their suggestions and comments on earlier versions of this paper.

## 1 Introduction

Given a weighted undirected graph  $G = (V, E, w)$  with non-negative edge weights given by  $w : E \rightarrow \mathbb{R}^+$ , and an integer  $k \in \mathbb{Z}^+$ , the DENSEST  $k$ -SUBGRAPH problem (DkS) asks to compute a set  $S \subset V$  of cardinality  $|S| \leq k$  such that the weight of edges inside  $S$  (i.e.,  $\sum_{i,j \in S} w(\{i,j\})$ ) is maximized (if  $\{i,j\} \notin E$ , we assume w.l.o.g. that  $w(\{i,j\}) = 0$ ). Computing the DkS of a graph is a fundamental NP-hard problem. There has been a lot of work on studying approximation algorithms for DkS, we give a brief survey in Section 1.3.

The current best known approximation algorithm [6] computes an  $\mathcal{O}(n^{1/4+\epsilon})$  approximation in time  $n^{\mathcal{O}(1/\epsilon)}$  for any  $\epsilon > 0$ . On the hardness side, Manurangsi [31] showed that assuming the exponential time hypothesis (ETH), there is no polynomial time algorithm that approximates this to within  $n^{1/(\log \log n)^c}$  factor where  $c > 0$  is some fixed constant. There are hardness of approximation results known for this problem assuming various other



© Yash Khanna and Anand Louis;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 27; pp. 27:1–27:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

hardness assumptions, see Section 1.3 for a brief survey. But there is still a huge gap between the upper and lower bounds on the approximability of this problem.

Given this status of the approximability of the DENSEST  $k$ -SUBGRAPH problem, we ask what are some “easier” instances of this problem? We propose some natural semi-random models of instances with a planted dense subgraph, and study approximation algorithms for computing the densest subgraph in them. Studying semi-random models of instances has been a very fruitful direction of study towards understanding the complexity for various NP-hard problems such as graph partitioning problems [28, 29, 26, 27], independent sets [13, 32], graph coloring [1, 11, 12], etc. By studying algorithms for instances where some parts are chosen to be arbitrary and some parts are chosen to be random, one can understand which aspects of the problem make it computationally intractable. Besides being of natural theoretical interest, studying approximation algorithms for semi-random models of instances can also be practically useful since some natural semi-random models of instances can be better models of instances arising in practice than the worst-case instances. Therefore, designing algorithms specifically for such models can help to bridge the gap between theory and practice in the study of algorithms. Some random and semi-random models of instances of the DENSEST  $k$ -SUBGRAPH problem (and its many variants) have been studied in [2, 6, 8, 19, 20, 21, 33, 34], we discuss them in Section 1.3. Our models are primarily inspired by the densest subgraph models mentioned above as well as the semi-random models of instances for other problems [13, 32] studied in the literature. For a large range of parameters of these models, we get significantly better approximation factors for the DENSEST  $k$ -SUBGRAPH problem, and also show that we can recover a large part of the planted solution.

We note that semidefinite programming (SDP) based methods have been popularly used in many randomized models for different problems, including the DENSEST  $k$ -SUBGRAPH problem [19, 20, 21]. And thus, another motivation for our work is to understand the power of SDPs in approximating the DENSEST  $k$ -SUBGRAPH problem. Since even strong SDP relaxations of the problem have a large integrality gap [7] for worst case instances (see Section 1.3), we ask what families of instances can SDPs approximate well? In addition to being of theoretical interest, algorithms using the basic SDP also have a smaller running time. In comparison, the algorithm of [6] produces an  $\mathcal{O}(n^{1/4+\epsilon})$  approximation for worst-case instances in time  $n^{\mathcal{O}(1/\epsilon)}$ ; their algorithm is based on rounding an LP hierarchy, but they also show that their algorithm can be executed without solving an LP and obtain the same guarantees.

## 1.1 Our models and results

The main inspiration for our models are the semi-random models of instances for the independent set problem [13, 32]. Their instances are constructed as follows. Starting with a set of vertices  $V$ , a subset of  $k$  vertices is chosen to form the independent set  $S$ , and edges are added between each pair in  $S \times (V \setminus S)$  independently with probability  $p$ . Finally, an arbitrary graph is added on  $V \setminus S$ . They study the values of  $k$  and  $p$  for which they can recover a large independent set. Our models can be viewed as analogs of this model to the DENSEST  $k$ -SUBGRAPH problem: edges are added between each pair in  $S \times (V \setminus S)$  independently with probability  $p$ , and then edges are added in  $S$  to form a dense subset. Since we also guarantee that we can recover a large part of the planted dense subgraph  $S$ , we also need to assume that the graph induced on  $V \setminus S$  is “far” from containing a dense subgraph. We now define our models.

► **Definition 1.1** ( $DkSEXP(n, k, d, \delta, d', \lambda)$ ). *An instance of  $DkSEXP(n, k, d, \delta, d', \lambda)$  is generated as follows,*

1. We partition  $V$  into two sets,  $S$  and  $V \setminus S$  with  $|S| = k$ . We add edges (of weight 1) between pairs in  $S \times (V \setminus S)$  independently with probability  $p \stackrel{\text{def}}{=} \delta d/k$ .
2. We add edges of arbitrary non-negative weights between arbitrary pairs of vertices in  $S$  such that the graph induced on  $S$  has average weighted degree  $d$ .
3. We add edges of arbitrary non-negative weights between arbitrary pairs of vertices in  $V \setminus S$  such that the graph induced on  $V \setminus S$  is a  $(d', \lambda)$ -expander (see Definition 1.10 for definition).
4. (Monotone adversary) Arbitrarily delete any of the edges added in step 1 and step 3.
5. Output the resulting graph.

We note that the step 2, step 3, and step 4 in the construction of the instance above are adversarial steps.

$DkSEXP(n, k, d, \delta, d', \lambda)$  are a class of instances that have a prominent dense subset of size  $k$ . Note that, since the graph induced on  $V \setminus S$  is a subset of an expander graph, it would not have any dense subsets. We also note that the monotone adversary can make significant changes to graph structure. For example, the graph induced on  $V \setminus S$  can be neither  $d'$ -regular nor an expander after the action of the monotone adversary.

We require  $\delta < 1$  in step 1 for the following reason. For any fixed set  $S' \subset V \setminus S$  such that  $|S'| = \mathcal{O}(k)$ , the expected weight of edges in the bipartite graph induced on  $S \cup S'$  is  $\mathcal{O}(\delta kd)$ . Since we want the graph induced on  $S$  to be the densest  $k$ -subgraph (the total of edges in the graph induced on  $S$  is  $kd/2$ ), we restrict  $\delta$  to be at most 1.

We present our main results below, note that our algorithm outputs a dense subgraph of size  $k$  and its performance is measured with respect to the density of the planted subgraph  $G[S]$ , i.e.  $kd/2$ .

► **Definition 1.2.** We define  $\rho(V') \stackrel{\text{def}}{=} \left( \sum_{i,j \in V'} w(\{i, j\}) \right) / 2$  for any  $V' \subseteq V$ .

► **Theorem 1.3** (Informal version of Theorem 2.1). Given an instance of  $DkSEXP(n, k, d, \delta, d', \lambda)$  where

$$\delta = \Theta\left(\frac{kd'}{nd}\right), \quad \frac{\delta d}{k} = \Omega\left(\frac{\log n}{n}\right), \quad \text{and} \quad \nu = \Theta\left(\sqrt{\delta + \frac{\lambda + \sqrt{d'}}{d}}\right),$$

there exists a deterministic polynomial time algorithm that outputs with high probability (over the instance) a vertex set  $\mathcal{Q}$  of size  $k$  such that  $\rho(\mathcal{Q}) \geq (1 - \nu) \frac{kd}{2}$ . The above algorithm also computes a vertex set  $T$  such that

$$(a) \quad |T| \leq (1 + \mathcal{O}(\nu))k. \quad (b) \quad \rho(T \cap S) \geq (1 - \mathcal{O}(\nu)) \frac{kd}{2}.$$

► **Remark 1.4.** In Theorem 1.3, we restrict the range of  $\delta$  for the following reason. An interesting setting of parameters is when the average degree of vertices in  $S$  and  $V \setminus S$  are within constant factors of each other. Then the expected average degree of a vertex in  $S$  is  $d + p(n - k)$ . And for a vertex in  $V \setminus S$ , the expected average degree is  $d' + kp$ . Thus setting,

$$d + p(n - k) = \Theta(d' + kp) \implies \delta = \Theta\left(\frac{kd'}{nd}\right) \quad \left(\text{Recall, } p = \frac{\delta d}{k}\right).$$

We also study another interesting model with a different assumption on the subgraph  $G[V \setminus S]$ .

## 27:4 Planted Models for the Densest $k$ -Subgraph Problem

► **Definition 1.5.**  $DkS(n, k, d, \delta, \gamma)$  is generated similarly to  $DkSEXP(n, k, d, \delta, d', \lambda)$  except in step 3, where we add edges between arbitrary pairs of vertices in  $V \setminus S$  such that the graph induced on  $V \setminus S$  has the following property:  $\rho(V') \leq \gamma d |V'| \quad \forall V' \subseteq V \setminus S$ .

By construction, the graph induced on  $V \setminus S$  does not have very dense subsets.

► **Theorem 1.6.** Given an instance of  $DkS(n, k, d, \delta, \gamma)$  where

$$\delta = \Theta\left(\frac{k}{n}\right), \quad \frac{\delta d}{k} = \Omega\left(\frac{\log n}{n}\right), \quad \text{and} \quad \tau = \Theta\left(\sqrt{\delta + \gamma + \frac{1}{\sqrt{d}}}\right),$$

there is a deterministic polynomial time algorithm that outputs with high probability (over the instance) a vertex set  $\mathcal{Q}$  of size  $k$  such that  $\rho(\mathcal{Q}) \geq (1 - \tau) \frac{kd}{2}$ . The above algorithm also computes a vertex set  $T$  such that

$$(a) \quad |T| \leq (1 + \mathcal{O}(\tau))k. \quad (b) \quad \rho(T \cap S) \geq (1 - \mathcal{O}(\tau)) \frac{kd}{2}.$$

### Other results

We also study two variants of  $DkSEXP(n, k, d, \delta, d', \lambda)$  and  $DkS(n, k, d, \delta, \gamma)$  where the subgraph  $G[S]$  is  $d$ -regular.

1.  $DkSEXPReg(n, k, d, \delta, d', \lambda)$  is same as  $DkSEXP(n, k, d, \delta, d', \lambda)$  except in step 2, which requires the subgraph  $G[S]$  to be an arbitrary  $d$ -regular graph.

► **Theorem 1.7.** Given an instance of  $DkSEXPReg(n, k, d, \delta, d', \lambda)$  where

$$\delta = \Theta\left(\frac{kd'}{nd}\right), \quad \frac{\delta d}{k} = \Omega\left(\frac{\log n}{n}\right), \quad \text{and} \quad \nu' = \Theta\left(\frac{\sqrt{d'}}{d\left(1 - \delta - \frac{\lambda}{d}\right)}\right),$$

there is a deterministic polynomial time algorithm that outputs with high probability (over the instance) a vertex set  $\mathcal{Q}$  of size  $k$  such that

$$a. \quad \rho(\mathcal{Q}) \geq (1 - \nu') \frac{kd}{2}. \quad b. \quad |\mathcal{Q} \cap S| \geq (1 - \mathcal{O}(\nu'))k.$$

2.  $DkSReg(n, k, d, \delta, \gamma)$  is same as  $DkS(n, k, d, \delta, \gamma)$  except in step 2, which requires the subgraph  $G[S]$  to be an arbitrary  $d$ -regular graph.

► **Theorem 1.8.** Given an instance of  $DkSReg(n, k, d, \delta, \gamma)$  where

$$\delta = \Theta\left(\frac{k}{n}\right), \quad \frac{\delta d}{k} = \Omega\left(\frac{\log n}{n}\right), \quad \text{and} \quad \tau' = \Theta\left(\frac{1}{\sqrt{d}(1 - \gamma - \delta)}\right),$$

there is a deterministic polynomial time algorithm that outputs with high probability (over the instance) a vertex set  $\mathcal{Q}$  of size  $k$  such that

$$a. \quad \rho(\mathcal{Q}) \geq (1 - \tau') \frac{kd}{2}. \quad b. \quad |\mathcal{Q} \cap S| \geq (1 - \mathcal{O}(\tau'))k.$$

We will show that for most natural regime of parameters, we get a better approximation factors in the case when  $G[S]$  is a  $d$ -regular graph.

► **Remark 1.9.** It has been pointed out to us by anonymous reviewers that for a large range of parameters of the  $DkS(n, k, d, \delta, \gamma)$  and  $DkSReg(n, k, d, \delta, \gamma)$  models,  $\arg \max_{W \subseteq V} \rho(W)/|W|$  will be a subset of  $S$ ; for any graph  $G = (V, E)$ , the algorithm due to Charikar [10] can be used to compute  $\arg \max_{W \subseteq V} \rho(W)/|W|$  in polynomial time. It is plausible that using this algorithm iteratively, one can recover a “large” part of  $S$ . However the algorithm described in Theorem 1.6 and Theorem 1.8 gives a more direct approach to recover a large part of  $S$ .

## 1.2 Notation

We use  $n \stackrel{\text{def}}{=} |V|$ , and use  $V$  and  $[n] \stackrel{\text{def}}{=} \{1, 2, \dots, n\}$  interchangeably. We assume w.l.o.g. that  $G$  is a complete graph: if  $\{i, j\} \notin E$ , we add  $\{i, j\}$  to  $E$  and set  $w(\{i, j\}) = 0$ . We use  $A$  to denote the weighted adjacency matrix of  $G$ , i.e.  $A_{ij} = w(\{i, j\}) \forall i, j \in V$ . The degree of vertex  $i$  is defined as  $d_i \stackrel{\text{def}}{=} \sum_{j \in V} w(\{i, j\})$ .

For  $V' \subseteq V$ , we use  $G[V']$  to denote the subgraph induced on  $V'$  and  $\overline{V'}$  to denote  $V \setminus V'$ . For a vector  $v$ , we use  $\|v\|$  to denote the  $\|v\|_2$ . For a matrix  $A$ , we use  $\|A\|$  to denote the spectral norm  $\|A\| \stackrel{\text{def}}{=} \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$ .

We define probability distributions  $\mu$  over finite sets  $\Omega$ . For a random variable (r.v.)  $X : \Omega \rightarrow \mathbb{R}$ , its expectation is denoted by  $\mathbb{E}_{\omega \sim \mu}[X]$ . In particular, we define the two distributions which we use below.

1. For a vertex set  $V' \subseteq V$ , we define a probability (uniform) distribution ( $f_{V'}$ ) on the vertex set  $V'$  as follows. For a vertex  $i \in V'$ ,  $f_{V'}(i) = \frac{1}{|V'|}$ . We use  $i \sim V'$  to denote  $i \sim f_{V'}$  for clarity.
2. For a vertex set  $V' \subseteq V$ , we define a probability distribution ( $f_{E(G[V'])}$ ) on the edges of  $G[V']$  as follows. For an edge  $e \in E(G[V'])$ ,  $f_{E(G[V'])}(e) = \frac{w(e)}{\rho(V')}$ . Again, we use  $e \sim E(G[V'])$  to denote  $e \sim f_{E(G[V'])}$  for convenience.

► **Definition 1.10** ( $(d, \lambda)$ -expanders). *A graph  $H = (V, E, w)$  is said to be a  $(d, \lambda)$ -expander if  $H$  is  $d$ -regular and  $|\lambda_i| \leq \lambda, \forall i \in [n] \setminus \{1\}$ , where  $\lambda_1 \geq \lambda_2 \dots \geq \lambda_n$  are the eigenvalues of the weighted adjacency matrix of  $H$ .*

## 1.3 Related Work

**Densest  $k$ -subgraph.** There has been a lot of work on the DENSEST  $k$ -SUBGRAPH problem and its variants. The current best known approximation algorithm, due to Bhaskara et al. [6], gives an approximation ratio of  $\mathcal{O}(n^{1/4+\epsilon})$  in time  $n^{\mathcal{O}(1/\epsilon)}$ , for all values of  $\epsilon > 0$  (for  $\epsilon = 1/\log n$ , we get a ratio of  $\mathcal{O}(n^{1/4})$ ). They also extend their approach to give a  $\mathcal{O}(n^{1/4-\epsilon})$  approximation algorithm which runs in time  $2^{n^{\mathcal{O}(\epsilon)}}$ . They improved the prior results of Feige et al. [14] which gave a  $n^{1/3-\epsilon}$  approximation for some small  $\epsilon > 0$ . [14] also give a greedy algorithm which has an approximation factor of  $\mathcal{O}(n/k)$ .

When  $k = \Theta(n)$ , Asahiro et al. [3] gave a constant factor approximation algorithm. Many other works have looked at this problem using linear and semidefinite programming techniques. Srivastav et al. [37] gave a randomized rounding algorithm using a SDP relaxation in the case when  $k = n/c$  for  $c > 1$ , they improved the constants for certain values of  $k$  over the results of [3]. Feige and Langberg [15] use a different SDP to get an approximation of slightly above  $k/n$  for the case when  $k$  is roughly  $n/2$ . Feige and Seltser [16] construct examples for which their SDP has an integrality gap of  $\Omega(n^{1/3})$ .

There has been work done on a related problem called the maximum density subgraph, where the objective is to find a subgraph which maximizes the ratio of number of edges to the number of vertices. Goldberg [18] and Gallo et al. [17] had given an algorithm to solve this problem exactly using maximum flow techniques. Later, Charikar [10] gave an algorithm based on a linear programming method. This paper also solves the problem for directed graphs using a notion of density given by Kannan and Vinay [22]. Khuller and Saha [24] gave a max-flow based algorithm in the directed setting.

On the hardness side, Khot [23] showed that it does not have a PTAS unless NP has subexponential algorithms. There has been some works based on some other hardness assumptions. Assuming the small-set expansion hypothesis, Raghavendra and Steurer [35] show that it is NP-hard to approximate DkS to any constant factor. Under the deterministic ETH assumption, Braverman et al. [9] show that it requires  $n^{\Omega(\log n)}$  time to approximate DkS with perfect completeness to within  $1 + \varepsilon$  factor (for a universal constant  $\varepsilon > 0$ ). More recently Manurangsi [31] showed assuming the exponential time hypothesis (ETH), that there is no polynomial time algorithm that approximates this to within  $n^{1/(\log \log n)^c}$  factor where  $c > 0$  is some fixed constant independent of  $n$ .

Bhaskara et al. [7] study strong SDP relaxations of the problem and show that the integrality gap of DkS remains  $n^{\Omega_\varepsilon(1)}$  even after  $n^{1-\varepsilon}$  rounds of the Lasserre hierarchy. Also for  $n^{\Omega(\varepsilon)}$  rounds, the gap is as large as  $n^{2/53-\varepsilon}$ . Moreover for the Sherali-Adams relaxation, they show a lower bound of  $\Omega(n^{1/4}/\log^3 n)$  on the integrality gap for  $\Omega(\log n/\log \log n)$  rounds.

Ames [2] studies the planted DkS problem using a non-SDP convex relaxation for instances of the following kind. Let  $S$  be the planted dense subgraph (of size  $k$ ), they claim that if  $G[S]$  contains at least  $\binom{k}{2} - c_1 k^2$  edges and the subgraph  $G[V \setminus S]$  contains at most  $c_2 k^2$  edges where  $c_1, c_2$  are constants depending on other parameters of the graph like the density of the subgraph  $G[S]$  etc, then under some mild technical conditions, they show that the unique optimal solution to their convex program is integral and corresponds to the set  $S$ . They also study analogous models for bipartite graphs.

**Random models for DkS.** Bhaskara et al. [6] study a few random models of instances for the DENSEST  $k$ -SUBGRAPH problem, we describe them here. Let  $\mathcal{D}_1$  denote the distribution of Erdős-Rényi random graphs  $G(n, p)$  and let  $\mathcal{D}_2$  denote the distribution of graphs constructed as follows. Starting with a “host graph” of average degree  $D$  ( $D \stackrel{\text{def}}{=} np$ ), a set  $S$  of  $k$  vertices is chosen arbitrarily and the subgraph on  $S$  is replaced with a dense subgraph of average degree  $d$ . Given  $G_1 \sim \mathcal{D}_1$  and  $G_2 \sim \mathcal{D}_2$ , the problem is to distinguish between the two distributions. They consider this problem in three different models with varying assumptions on  $\mathcal{D}_2$ , (i) *Random Planted Model* : the host graph and the planted dense subgraph are random, (ii) *Dense in Random Model* : an arbitrary dense graph is planted inside a random graph, and (iii) *Dense vs Random Model* : an arbitrary dense graph is planted inside an arbitrary graph.

The *planted dense subgraph* recovery problem is similar in spirit to the *Random Planted Model* where the goal is to recover a hidden community of size  $k$  within a larger graph which is constructed as follows : two vertices are connected by an edge with probability  $p$  if they belong to the same community and with probability  $q$  otherwise. The typical setting of parameters is,  $p > q$ . The works by [33, 20, 34, 19, 21, 8, 2] studies this problem using SDP based, spectral, statistical, message passing algorithms etc.

We give a brief overview of their distinguishing algorithms in the three models. Given a graph on  $n$  vertices with average degree  $d_{\text{avg}}$ , its log-density is defined as  $\frac{\log d_{\text{avg}}}{\log n}$ . Let

$\Theta_1$  and  $\Theta_2$  denote the log-density of  $G_1$  and the log-density of the planted subgraph  $G_2[S]$  respectively. Their algorithm is based on the counts of a specially constructed small-sized tree (the size of which is parameterized by relatively prime integers  $r, s$  such that  $s > r > 0$ ) as a subgraph in  $G_1$  and  $G_2$ . They show that if  $\Theta_1 \leq r/s$ , then  $G_1$  will have at most poly-logarithmic ( $\mathcal{O}(\log n)^{s-r}$ ) number of such subtrees. On the other hand, when  $\Theta_2 \geq r/s + \epsilon$  where  $\epsilon > 0$  is a small constant, they show that there at least  $k^\epsilon$  such subtrees (even in the *Dense vs Random Model*). Now if  $k > (\log n)^{\omega(1)}$ , they use this difference in the log-densities to show the gap between counts of such trees in  $G_1$  and  $G_2$ , and hence are able to distinguish between the two distributions. They show that the running time of this algorithm is  $n^{\mathcal{O}(r)}$ . Also for constant  $\Theta_1$  and  $\Theta_2$ , the running time is  $n^{\mathcal{O}(1/(\Theta_2 - \Theta_1))}$  ([6, 5]). We call this algorithm the “subgraph counting” algorithm.

The distinguishing problem can be restated as the following : For a given  $n, k, p$ , we are interested in finding the smallest value of  $d$  for which the problem can be solved. For a certain range of parameters, spectral, SDP based methods, etc. can be used to work for small values of  $d$ . For example, in the *Dense vs Random Model*, when  $k > \sqrt{n}$  a natural SDP relaxation of  $DkS$  can be used to distinguish between  $G_1$  and  $G_2$  for  $d > \sqrt{D} + kD/n$  (which is smaller than  $D^{\log_n k}$ , the threshold of the subgraph counting algorithm). They upper bound the cost of the optimal SDP solution for a random graph  $G_1$ , by constructing a feasible dual solution which certifies (w.h.p.) that it cannot contain a  $k$ -subgraph with density more than that of  $\sqrt{D} + kD/n$ . We use their results in bounding the cost of the SDP contribution from  $G[V \setminus S]$  in the  $DkSExp(n, k, d, \delta, d', \lambda)$  and  $DkSExpReg(n, k, d, \delta, d', \lambda)$  models.

The distribution  $\mathcal{D}_2$  of graphs considered in the *Dense in Random Model* (arbitrary dense graph planted in a random graph) is similar to a subset of  $DkSExp(n, k, d, \delta, d', \lambda)$  instances since  $G[S]$  is an arbitrary dense subgraph in both models and  $G[S, V \setminus S]$  is a random graph in both the models. The difference is in the subgraph  $G[V \setminus S]$ , where this is a random graph in the *Dense in Random* model whereas our models require it to be a regular expander. While our proofs require the expander to be regular, they can also be made to work for random graphs since we use the bound on the SDP value from [6] (analysis in Section 2.2). We note that while random graphs are good expanders w.h.p., the converse of this fact is not true in general, since there are known deterministic constructions of expander graphs.

We look at the range of parameters where the following two algorithms can be used to solve the *Dense in Random* problem. One is the SDP based algorithm proposed in our work (closely related to  $DkSExp(n, k, d, \delta, d', \lambda)$  model) and second is the subgraph counting algorithm which uses the difference in the log-densities of the planted subgraph and the host graph to distinguish the two distributions from [6, 5]. For the purposes of comparison, we consider the case when  $k, d = poly(n)$  and  $p = 1/poly(n)$ . Also we ignore the low-order terms in these expressions. In this regime, our algorithms’ threshold is

$$d = \Omega(\max\{pk, \sqrt{np}\}) \quad (1)$$

since we can use the objective value of the SDP 1.11 to distinguish between the cases in this range of  $d$ . For  $G_1$ , this value is at most  $k(pk + \sqrt{np})/2$  (Lemma 2.12) while for  $G_2$  it is at least  $kd/2$ . Moreover, Algorithm 1 can be used to recover a part of the planted solution as the value of  $\nu$  is small (when  $d$  satisfies Equation (1),  $\nu$  is bounded away and smaller than 1) in this regime (see Section 2 and Theorem 2.1).

The counting algorithms’ threshold (or the log-density threshold) is

$$\frac{\log d}{\log k} - \frac{\log np}{\log n} > 0 \iff \log d > \frac{\log k \log np}{\log n} \iff d = \Omega((np)^{\log_n k})$$

and its running time is  $n^{\mathcal{O}\left(\frac{1}{\log_k d - \log_n np}\right)}$ . We look at different ranges of  $k$  and compare the values of  $d$  for which the two algorithms can solve the distinguishing problem.

1.  $k = \Theta(\sqrt{n})$ .

In this case,  $\max\{pk, \sqrt{np}\} = \sqrt{np}$ . This matches with the log-density threshold. Note that for  $p = \Theta(1/\sqrt{n})$ , we get  $d = \Omega(n^{1/4})$ . To the best of our knowledge, there is no poly-time algorithm which beats this lower bound.

2.  $k = \omega(\sqrt{n})$ .

In this setting,  $(np)^{\log_n k} = \omega(\sqrt{np})$ . Also,  $(np)^{\log_n k} = k(p)^{\log_n k} = \omega(pk)$ . Thus our algorithm has a better threshold in this regime. There is a spectral algorithm, see Section 6.2 of [6], which uses the second eigenvalue of the adjacency matrix which can distinguish with the same threshold as our algorithm in this regime.

3.  $k = o(\sqrt{n})$ .

In this case,  $(np)^{\log_n k} = o(\sqrt{np})$ . Here the log-density threshold is smaller than our threshold. Therefore the algorithm by Bhaskara et al. [6] works for a larger range of parameters than our algorithms.

**Other semi-random models.** Semi-random instances of many other fundamental problems have been studied in the literature. This includes the unique games problem [25], graph coloring [1, 11, 12], graph partitioning problems such as balanced-cut, multi-cut, small set expansion [28, 29, 26, 27], etc. [30] studies the problem of learning communities in the Stochastic Block Model in the presence of adversarial errors.

McKenzie, Mehta and Trevisan [32] study the complexity of the independent set problem in the Feige-Killian model [13]. Instead of using a SDP relaxation for the problem, they use a “crude” SDP (introduced in [25]) which exploits the geometry of vectors (orthogonality etc.) to reveal the planted set. They bound the SDP contribution by the vertex pairs,  $S \times V \setminus S$  using the Grothendieck inequality and thereby showing that the vectors in  $S$  are “clustered” together. Their algorithm outputs w.h.p. a large independent set when  $k = \Omega(n^{2/3}/p^{1/3})$ . Also, for the parameter range  $k = \Omega(n^{2/3}/p)$ , it outputs a list of at most  $n$  independent sets of size  $k$ , one of which is the planted one.

**Semi-random models for graph partitioning problems.** The problem of  $DkS$  is very closely related to the SMALL SET EXPANSION problem (SSE, henceforth). This problem has been very well studied in the literature. At the first glance, the problem of  $DkS$  can be thought of as finding a small set  $S$  of size  $k$  which is non-expanding. The densest set is typically a non-expanding set because most of the edges incident on  $S$  would remain inside it than leaving it. But the converse is not true, since all sets of cardinality  $k$  which have small expansion are not dense. In particular, in our model, by the action of the monotone adversary on  $V \setminus S$ , there can exist many small sets (of size  $\mathcal{O}(k)$ ) which not only have a very small fraction of edges going outside but can have very few edges left inside as well. This makes the problem of  $DkS$  very different from the SSE problem. Nevertheless, we survey some related works of semi-random models of SSE. The works [36, 4] study the worst-case approximation factors for the SSE problem and give bi-criteria approximation algorithms for the same. Their algorithms are also based on rounding a SDP relaxation.

Makarychev, Markarychev and Vijayaraghavan [28] study the complexity of many graph partitioning problems including balanced cut, SSE, and multi-cut etc. They consider the following model : Partition  $V$  into  $(S, V \setminus S)$  such that  $G[S]$  and  $G[V \setminus S]$  are arbitrary while

$G[S, V \setminus S]$  is a random graph with some probability  $\varepsilon$ . They allow an adversary to add edges within  $S$  and  $V \setminus S$ , and delete any edges across these sets. They get constant factor bi-criteria approximation algorithms (under some mild technical conditions) in this model. In the case of balanced cut and SSE problems, when the partitions themselves have enough expansion within them, they can recover the planted cut upto a small error.

Louis and Venkat [26] study the problem of balanced vertex expansion in a natural semi-random model and get a bi-criteria approximation algorithm for the same. They even get an exact recovery for a restricted set of parameters in their model. Their proof consisted of constructing an optimal solution to the dual of the SDP relaxation and using it to show the integrality of the optimal primal solution. In [27], they study the problem for a general, balanced  $k$ -way vertex (and edge) expansion and give efficient algorithms for the same. Their construction consists of  $k$  (almost) regular expander graphs (over vertices  $\{S_i\}_{i=1}^k$ , each of size  $n/k$ ) and then adding edges across them ensuring that the expansion of each of the  $G[S_i]$ 's is small. Their algorithm is based on rounding a SDP relaxation and then showing that the vertices of each  $S_i$  are “clustered” together around the mean vector  $\mu_i$  and for different sets  $S_i$  and  $S_j$ ,  $\mu_i$  and  $\mu_j$  are sufficiently apart. This gives a way to recover a good solution. Our approach also shows that the SDP vectors for the vertices in  $S$  are “clustered” together. However arriving at such a conclusion requires different ideas because of the new challenges posed by the nature of the problem and assumptions on our models.

## 1.4 SDP formulation

We use the following Semidefinite/Vector Programming relaxation for our problem, over the vectors  $X_i$  ( $i \in [n]$ ) and  $I$ .

### ► SDP 1.11.

$$\text{maximize} \quad \frac{1}{2} \sum_{i,j=1}^n A_{ij} \langle X_i, X_j \rangle \quad (2)$$

$$\text{subject to} \quad \sum_{i=1}^n \langle X_i, X_i \rangle = k \quad (3)$$

$$\sum_{j=1}^n \langle X_i, X_j \rangle \leq k \langle X_i, X_i \rangle \quad \forall i \in [n] \quad (4)$$

$$0 \leq \langle X_i, X_j \rangle \leq \langle X_i, X_i \rangle \quad \forall i, j \in [n], (i \neq j) \quad (5)$$

$$\langle X_i, X_i \rangle \leq 1 \quad \forall i \in [n] \quad (6)$$

$$\langle X_i, I \rangle = \langle X_i, X_i \rangle \quad \forall i \in [n] \quad (7)$$

$$\langle I, I \rangle = 1 \quad (8)$$

We note that these programs can be solved efficiently using standard algorithms, like ellipsoid and interior point methods. To see, why the above SDP 1.11 is a relaxation, let  $S$  be the optimal set and  $v$  be any unit vector. It is easy to verify the solution set,

$$X_i = \begin{cases} v & i \in S \\ 0 & i \in V \setminus S \end{cases} \quad \text{and} \quad I = v.$$

is feasible for SDP 1.11 and gives the objective value equal to its optimal density.

## 1.5 Proof Overview

Our algorithms are based on rounding an SDP relaxation (SDP 1.11) for the DENSEST  $k$ -SUBGRAPH problem. At a high level, we show that most of the SDP mass is concentrated on the vertices in  $S$  (Proposition 2.16). To show this, we begin by observing that the SDP objective value is at least  $kd/2$  since the integer optimal solution to the SDP has value at least  $kd/2$ . Therefore, by proving an appropriate upper bound on the SDP value from edges in  $S \times (V \setminus S)$  (Proposition 2.2) and the edges in  $V \setminus S$  (Proposition 2.11), we can get a lower bound on the SDP value from the edges inside  $S$ .

The edges in  $S \times (V \setminus S)$  form a random bipartite graph. We can bound the contribution towards the SDP mass from this part by bounding the contribution from the “expected graph” (Lemma 2.5) and the contribution from the random graph minus the expected graph (Corollary 2.10). The contribution from the latter part can be bounded using bounds on the spectra of random matrices (Corollary 2.8). Since the expected graph is a complete weighted graph with edge weights equal to the edge probability, the contribution from this part can be bounded using the SDP constraints (Lemma 2.5).

For  $DkSEXP(n, k, d, \delta, d', \lambda)$  and  $DkSExpReg(n, k, d, \delta, d', \lambda)$ , we use a result by [6]. They construct a feasible solution to the dual of the SDP for random graphs, thereby bounding the cost of the optimal solution of the primal. Their proof only uses a bound on the spectral gap of the graph, and therefore, holds also for expander graphs. Therefore, this result gives us the desired bound on the SDP value on the edges inside  $V \setminus S$  in these models (Proposition 2.11). We also give an alternate proof of the same result using the spectral properties of the adjacency matrix of  $V \setminus S$  in the full version of the paper; this approach is similar in spirit to the proof of the classical *expander mixing lemma*.

For  $DkS(n, k, d, \delta, \gamma)$  and  $DkSReg(n, k, d, \delta, \gamma)$ , we bound the SDP value on the edges inside  $V \setminus S$  using a result of Charikar [10]. This work showed that for a graph  $H = (V', E')$ , a natural LP relaxation can be used to compute  $\max_{W \subseteq V'} \rho(W)/|W|$ . We show that we can use our SDP solution to construct a feasible solution for this LP. Since  $\rho(W)/|W| \leq \gamma d$ ,  $\forall W \subseteq V \setminus S$  in this model, Charikar’s result [10] implies that the cost of any feasible LP solution can be bounded by  $\gamma d$ . This gives us the desired bound on the SDP value on the edges inside  $V \setminus S$  in these models.

These bounds establish that most of the SDP mass is on the edges inside  $S$ . Using the SDP constraints, we show that the set of vertices corresponding to all the “long” vectors will contain a large weight of edges inside  $S$  (Corollary 2.19). Moreover, since the sum of squared lengths of the vectors is  $k$  (from the SDP constraints), we can only have  $\mathcal{O}(k)$  long vectors (Lemma 2.20). Using standard techniques from the literature, we can prune this set to obtain a set of size at most  $k$  and having large density [37]. In the case when the graph induced on  $S$  is  $d$ -regular, we show that if a set contains a large fraction of the edges inside  $S$ , then it must also have a large intersection with  $S$ . We present our complete procedure in Algorithm 1.

We note that while this framework for showing that the SDP mass is concentrated on the planted solution has been used for designing algorithms for semi-random instances of other problems as well, proving quantitative bounds is problem-specific and model-specific: different problems and different models require different approaches.

### Organization of the paper

Due to space constraints, we present the complete version (with all the details and proofs) of Section 2 in the full version of the paper, however we do state the key technical results

here with the proof of Theorem 2.1. We state and prove the formal versions of Theorem 1.6, Theorem 1.7, and Theorem 1.8 in the full version of the paper.

## 2 Analysis of $DkSEXP(n, k, d, \delta, d', \lambda)$

In this section, we will analyse the  $DkSEXP(n, k, d, \delta, d', \lambda)$  model. Our main result is the following.

► **Theorem 2.1** (Formal version of Theorem 1.3). *There exist universal constants  $\kappa, \xi \in \mathbb{R}^+$  and a deterministic polynomial time algorithm, which takes an instance of  $DkSEXP(n, k, d, \delta, d', \lambda)$  where*

$$\nu = 2\sqrt{3\left(6\delta + \xi\sqrt{\frac{\delta n}{dk}} + \frac{\lambda}{d} + \frac{d'k}{(n-k)d}\right)},$$

satisfying  $\nu \in (0, 1)$ , and  $\delta d/k \in [\kappa \log n/n, 1)$ , and outputs with high probability (over the instance) a vertex set  $\mathcal{Q}$  of size  $k$  such that

$$\rho(\mathcal{Q}) \geq (1 - \nu) \frac{kd}{2}.$$

The above algorithm also computes a vertex set  $T$  such that

$$(a) |T| \leq k \left(1 + \frac{\nu}{5}\right). \quad (b) \rho(T \cap S) \geq \left(1 - \frac{\nu}{2}\right) \frac{kd}{2}.$$

In the analysis below, without loss of generality we can ignore the adversarial action (step 4 of the model construction) to have taken place. Let us assume the monotone adversary removes edges arbitrarily from the subgraphs  $G[V \setminus S]$  &  $G[S, V \setminus S]$  and the new resulting adjacency matrix is  $A'$ . Then for any feasible solution  $\{\{Y_i\}_{i=1}^n, I\}$  of the SDP, we have  $\sum_{i \in P, j \in Q} A'_{ij} \langle Y_i, Y_j \rangle \leq \sum_{i \in P, j \in Q} A_{ij} \langle Y_i, Y_j \rangle$  for  $\forall P, Q \subseteq V$ . This holds because of the non-negativity constraint Equation (5). Thus the upper bounds on SDP contribution by vectors in  $G[S, V \setminus S]$  and  $G[V \setminus S]$  as claimed by Proposition 2.2 and Proposition 2.11 respectively are intact and the rest of the proof follows exactly. Hence, without loss of generality, we can ignore this step in the analysis of our algorithm.

### 2.1 Edges between $S$ and $V \setminus S$

In this section, we show an upper bound on  $\sum_{i \in S, j \in V \setminus S} A_{ij} \langle X_i, X_j \rangle$ .

► **Proposition 2.2.** *W.h.p. (over the choice of the graph), we have*

$$\sum_{i \in S, j \in V \setminus S} A_{ij} \langle X_i, X_j \rangle \leq 3pk^2 \left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right) + \xi k \sqrt{np} \sqrt{\left(\mathbb{E}_{i \sim S} \|X_i\|^2\right) \left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right)}.$$

Note that

$$\sum_{i \in S, j \in V \setminus S} A_{ij} \langle X_i, X_j \rangle = p \sum_{i \in S, j \in V \setminus S} \langle X_i, X_j \rangle + \sum_{i \in S, j \in V \setminus S} (A_{ij} - p) \langle X_i, X_j \rangle. \quad (9)$$

We will bound the two terms in the R.H.S. of Equation (9) separately. The first term relies only on the *expected graph* and can be bounded using the SDP constraints. We use bounds on the eigenvalues of random bipartite graphs to bound the second term.

### Bound the contribution from the *expected graph*

We first prove some properties of the SDP solutions that we will use to bound this term. The following lemma shows that if the expected value of the squared norm of the vectors corresponding to the set  $S$  is “large”, then their expected pairwise inner product is “large” as well.

► **Lemma 2.3.** *Let  $\{\{Y_i\}_{i=1}^n, I\}$  be any feasible solution of SDP 1.11 and  $T \subseteq V$  such that,  $\mathbb{E}_{i \sim T} \|Y_i\|^2 \geq 1 - \epsilon$  where  $0 \leq \epsilon \leq 1$ , then  $\mathbb{E}_{i, j \sim T} \langle Y_i, Y_j \rangle \geq 1 - 4\epsilon$ .*

► **Corollary 2.4.**

$$\mathbb{E}_{i, j \sim S} \langle X_i, X_j \rangle \geq 4 \mathbb{E}_{i \sim S} \|X_i\|^2 - 3.$$

We are now ready to bound the first term in Equation (9).

► **Lemma 2.5.**

$$\sum_{i \in S, j \in V \setminus S} \langle X_i, X_j \rangle \leq 3k^2 \left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right).$$

### Bounding the deviation from the *expected graph*

We now prove the following lemmas which we will use to bound the second term in Equation (9). Let  $B$  be the  $n \times n$  matrix defined as follows.

$$B_{ij} \stackrel{\text{def}}{=} \begin{cases} A_{ij} - p & i \in S, j \in V \setminus S \text{ or } i \in V \setminus S, j \in S \\ 0 & \text{otherwise} \end{cases}.$$

► **Lemma 2.6.**

$$\sum_{i, j \in V} B_{ij} \langle X_i, X_j \rangle \leq 2k \|B\| \sqrt{\left(\mathbb{E}_{i \sim S} \|X_i\|^2\right) \left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right)}.$$

Now, we use the following folklore result to bound  $\|B\|$ .

► **Theorem 2.7** ([21], Lemma 30). *Let  $M$  be a symmetric matrix of size  $n \times n$  with zero diagonals and independent entries such that  $M_{ij} = M_{ji} \sim \text{Bern}(p_{ij})$  for all  $i < j$  with  $p_{ij} \in [0, 1]$ . Assume  $p_{ij}(1 - p_{ij}) \leq r$  for all  $i < j$  and  $nr = \Omega(\log n)$ . Then, with high probability (over the randomness of matrix  $M$ ),*

$$\|M - \mathbb{E}[M]\| \leq \mathcal{O}(1)\sqrt{nr}.$$

► **Corollary 2.8.** *There exists universal constants  $\kappa, \xi \in \mathbb{R}^+$  such that if  $p \in \left[\frac{\kappa \log n}{n}, 1\right)$ , then*

$$\|B\| \leq \xi \sqrt{np}$$

*with high probability (over the choice of the graph).*

► **Remark 2.9.** Note that, Corollary 2.8 holds with high probability when  $p = \Omega(\log n/n)$ . In the rest of the paper, we work in the range of parameters where this lower bound on  $p$  is satisfied. However, we do restate it when explicitly using this bound.

► **Corollary 2.10.** *W.h.p. (over the choice of the graph),*

$$\sum_{i, j \in V} B_{ij} \langle X_i, X_j \rangle \leq 2\xi k \sqrt{np} \sqrt{\left(\mathbb{E}_{i \sim S} \|X_i\|^2\right) \left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right)}.$$

## 2.2 Edges in $V \setminus S$

We recall, the subgraph  $G[V \setminus S]$  is a  $(d', \lambda)$ -expander in the  $DkSEXP(n, k, d, \delta, d', \lambda)$  model. We show the following upper bound on the SDP mass contribution by the vectors in  $V \setminus S$ .

► **Proposition 2.11.**

$$\sum_{i,j \in V \setminus S} A_{ij} \langle X_i, X_j \rangle \leq \left( \lambda k + \frac{d' k^2}{n - k} \right) \left( 1 - \mathbb{E}_{i \sim S} \|X_i\|^2 \right).$$

To prove the above proposition, we use the following results from the Bhaskara et al. [6] paper.

► **Lemma 2.12** ([6], Theorem 6.1). *For a  $G(n, p)$  (Erdős-Rényi model) graph, the value of the SDP (SDP 1.11) is at most  $k^2 p + \mathcal{O}(k\sqrt{np})$  with high probability when  $p = \Omega(\log n/n)$ .*

► **Lemma 2.13** ([6], Theorem 6.1). *For a  $(d', \lambda)$ -expander graph on  $n$  vertices, the value of the SDP (SDP 1.11) is at most  $\frac{k^2 d'}{n} + k\lambda$ .*

We note that, though the statement proved in [6] is about random graphs (Lemma 2.12), their proof follows as is for an expander graph. Since, we are only applying Lemma 2.13 to the subgraph  $G[V \setminus S]$ , we use a scaling factor of  $\left(1 - \mathbb{E}_{i \sim S} \|X_i\|^2\right)$ . The proof of Proposition 2.11 follows directly from the above lemma. We also provide an alternate proof of this in the full version of the paper.

► **Remark 2.14.** If the subgraph,  $G[V \setminus S]$  is a random graph ( $G(n - k, p)$ ) as considered in our discussion in Section 1.3, we can analogously use Lemma 2.12 to get upper bounds on  $\sum_{i,j \in V \setminus S} A_{ij} \langle X_i, X_j \rangle$ .

## 2.3 Putting things together

We have shown upper bounds on the SDP mass from the edges in  $S \times (V \setminus S)$  (Proposition 2.2) and from the edges in  $V \setminus S$  (Proposition 2.11). We combine these results to show that the average value of  $\langle X_u, X_v \rangle$  where  $\{u, v\} \in E(G[S])$  is “large” (Proposition 2.16). The SDP constraint Equation (5) implies the corresponding vertices,  $u$  and  $v$  have large squared norms as well. This immediately guides us towards a selection criteria/recovery algorithm. However we need to output a vertex set of size at most  $k$ , we prune this set using a greedy strategy (Algorithm 1).

► **Lemma 2.15.**

$$\sum_{i,j \in S} A_{ij} \langle X_i, X_j \rangle = (kd) \mathbb{E}_{\{i,j\} \sim E(G[S])} \langle X_i, X_j \rangle.$$

► **Proposition 2.16.** *W.h.p. (over the choice of the graph), we have  $\mathbb{E}_{\{i,j\} \sim E(G[S])} \langle X_i, X_j \rangle \geq 1 - \eta$ , where*

$$\eta = 6\delta + \xi \sqrt{\frac{\delta n}{dk}} + \frac{\lambda}{d} + \frac{d' k}{(n - k)d}.$$

Now, we present the complete algorithm below.

## 27:14 Planted Models for the Densest $k$ -Subgraph Problem

■ **Algorithm 1** Recovering a dense set  $\mathcal{Q}$ .

---

**Input:** An Instance of  $\text{DkSEXP}(n, k, d, \delta, d', \lambda)$  /  $\text{DkSEXPReg}(n, k, d, \delta, d', \lambda)$  /  $\text{DkS}(n, k, d, \delta, \gamma)$  /  $\text{DkSReg}(n, k, d, \delta, \gamma)$  and a parameter  $0 < \eta < 1$ .

**Output:** A vertex set  $\mathcal{Q}$  of size  $k$ .

- 1: Solve SDP 1.11 to get the vectors  $\{\{X_i\}_{i=1}^n, I\}$ .
- 2:  $\alpha = \begin{cases} 1/\sqrt{3\eta} & \text{For instances of type, DkSEXP}(n, k, d, \delta, d', \lambda) \text{ or DkS}(n, k, d, \delta, \gamma) . \\ 2/\sqrt{\eta} & \text{For instances of type, DkSEXPReg}(n, k, d, \delta, d', \lambda) \text{ or DkSReg}(n, k, d, \delta, \gamma). \end{cases}$
- 3: Let  $T = \{i \in V : \|X_i\|^2 \geq 1 - \alpha\eta\}$ .
- 4: Initialize  $\mathcal{Q} = T$ .
- 5: **if**  $|\mathcal{Q}| < k$  **then**
- 6:     Arbitrarily add remaining vertices to set  $\mathcal{Q}$  to make its size  $k$ .
- 7: **else**
- 8:     **while**  $|\mathcal{Q}| \neq k$  **do**
- 9:         Remove the minimum weighted vertex from the set  $\mathcal{Q}$ .
- 10:     **end while**
- 11: **end if**
- 12: Return  $\mathcal{Q}$ .

---

Note that if  $\eta = 0$ , the SDP returns an integral solution and we can recover the set  $S$  exactly. Therefore, w.l.o.g. we assume  $\eta \neq 0, 1$ .

To analyse the cost of the solution returned by Algorithm 1, we define two sets as follows.

$$T' \stackrel{\text{def}}{=} \{\{i, j\} \in E : \langle X_i, X_j \rangle \geq 1 - \alpha\eta\} \quad \text{and} \quad T \stackrel{\text{def}}{=} \{i \in V : \|X_i\|^2 \geq 1 - \alpha\eta\},$$

where  $1 < \alpha < 1/\eta$  is a parameter to be fixed later.

We show that a *large* weight of the edges inside  $S$  also lies in the set  $T'$ .

► **Lemma 2.17.** *W.h.p. (over the choice of the graph),*

$$\sum_{e \in T' \cap E(G[S])} w(e) \geq \frac{kd}{2} \left(1 - \frac{1}{\alpha}\right).$$

The following lemma shows that the subgraph induced on  $T \cap S$  contains all the edges in  $T' \cap E(G[S])$ .

► **Lemma 2.18.** *W.h.p. (over the choice of the graph),*

$$T' \cap E(G[S]) \subseteq E(G[T \cap S]).$$

► **Corollary 2.19.** *W.h.p. (over the choice of the graph),*

$$\rho(T) \geq \rho(T \cap S) \geq \frac{kd}{2} \left(1 - \frac{1}{\alpha}\right).$$

We have shown that the subgraph induced on  $T$  has a large weight ( $\approx kd/2$ ). In the next lemma, we show that the size of set  $T$  is not too large compared to  $k$ .

► **Lemma 2.20.** *W.h.p. (over the choice of the graph),*

$$|T| \leq \frac{k}{1 - \alpha\eta}.$$

To prune the set  $T$  and obtain a set of size  $k$ , we use a lemma from the work by Srivastav et al. [37].

► **Lemma 2.21** ([37], Lemma 1). *Let  $V', V'' \subseteq V$  be non-empty subsets such that  $|V''| \geq |V'|$ , then the greedy procedure which picks the lowest weighted vertex from  $V''$  and removes it iteratively till we have  $|V'|$  vertices left ensures,  $\rho(V') \geq \frac{|V'|(|V'| - 1)}{|V''|(|V''| - 1)} \rho(V'')$ .*

We are now ready to prove the main result which gives the approximation guarantee of our algorithm. We also set the value of parameter  $\alpha$  which maximizes the density of the output graph.

**Proof of Theorem 2.1.** We run Algorithm 1 on  $\text{DkSEXP}(n, k, d, \delta, d', \lambda)$  with  $\eta$  as given in Proposition 2.16. From Lemma 2.21, we have a handle on the density of the new set ( $\mathcal{Q}$ ) after pruning  $T$  to a set of size  $k$ . The algorithm performs this exactly in the steps 5 to 11. Let  $\text{ALG}$  denote the density of this new set (output of Algorithm 1). We have,

$$\begin{aligned} \text{ALG} &\geq \left( \frac{k(k-1)}{|T|(|T|-1)} \right) \left( 1 - \frac{1}{\alpha} \right) \frac{kd}{2} \quad (\text{by Corollary 2.19 and Lemma 2.21}) \\ &\geq \left( \frac{(1-\alpha\eta)^2}{1+\alpha\eta/(k-1)} \right) \left( 1 - \frac{1}{\alpha} \right) \frac{kd}{2} \quad (\text{by Lemma 2.20 and dividing by } k-1) \\ &\geq \left( \frac{(1-\alpha\eta)^2}{1+\alpha\eta} \right) \left( 1 - \frac{1}{\alpha} \right) \frac{kd}{2} \quad (\text{w.l.o.g., } k \geq 2) \\ &\geq (1-2\alpha\eta)(1-\alpha\eta) \left( 1 - \frac{1}{\alpha} \right) \frac{kd}{2} \quad \left( (1-x)^2 \geq 1-2x \ \& \ \frac{1}{1+x} \geq 1-x, \ \forall x \in \mathbb{R}_{\geq 0} \right) \\ &\geq \left( 1 - 3\alpha\eta - \frac{1}{\alpha} \right) \frac{kd}{2} \quad (\text{rearranging and bounding the positive terms by } 0) \\ &= \left( 1 - 2\sqrt{3\eta} \right) \frac{kd}{2} \quad \left( \text{we fix } \alpha = 1/\sqrt{3\eta} \right). \end{aligned}$$

Letting  $\nu \stackrel{\text{def}}{=} 2\sqrt{3\eta}$ , we get that  $\text{ALG} \geq (1-\tau)kd/2$  where

$$\nu = 2\sqrt{3 \left( 6\delta + \xi\sqrt{\frac{\delta n}{dk}} + \frac{\lambda}{d} + \frac{d'k}{(n-k)d} \right)} \quad (\text{using the value of } \eta \text{ from Proposition 2.16}).$$

From Lemma 2.20,  $|T| \leq \frac{k}{1-\alpha\eta} = \frac{k}{1-(\nu/6)} \leq k \left( 1 + \frac{\nu}{5} \right)$ . And from Corollary 2.19,  $\rho(T \cap S) \geq \frac{kd}{2} \left( 1 - \frac{1}{\alpha} \right) = \frac{kd}{2} \left( 1 - \frac{\nu}{2} \right)$ . ◀

Note that for the parameter range  $0 < 2\sqrt{3\eta} < 1 \iff 0 < \nu < 1$ , the value of  $\alpha (= 1/\sqrt{3\eta})$  fixed by the algorithm lies in the interval  $(1, 1/\eta)$  as required.

► **Remark 2.22** (on Theorem 1.3). In the restricted parameter case, we simplify the arguments in our informal theorem statements, i.e. the case when the average degree of vertices in  $S$  and  $V \setminus S$  is close, we have  $\delta = \Theta \left( \frac{kd'}{nd} \right)$ . Assuming  $\nu = 2\sqrt{3\eta}$ , we rewrite  $\frac{\delta n}{dk}$  as  $\frac{d'}{d^2}$  from the above value of  $\delta$  and the term  $\frac{(d'-\lambda)k}{(n-k)d}$  is at most a constant for “large”  $n$ . So, the new value of  $\tau$  is  $\Theta \left( \sqrt{\delta + \frac{\lambda + \sqrt{d'}}{d}} \right)$ . A similar argument gives the new value of  $\nu'$  in Theorem 1.7.

## References

- 1 Noga Alon and Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM J. Comput.*, 26(6):1733–1748, December 1997. doi:10.1137/S0097539794270248.
- 2 Brendan P. Ames. Guaranteed recovery of planted cliques and dense subgraphs by convex relaxation. *J. Optim. Theory Appl.*, 167(2):653–675, November 2015. doi:10.1007/s10957-015-0777-x.
- 3 Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. In *Algorithm Theory — SWAT’96*, pages 136–148, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- 4 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM J. Comput.*, 43(2):872–904, 2014. doi:10.1137/120873996.
- 5 Aditya Bhaskara. *Finding dense structures in graphs and matrices*. PhD thesis, Princeton University, 2012. URL: <https://www.cs.utah.edu/~bhaskara/files/thesis.pdf>.
- 6 Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an  $O(n^{1/4})$  approximation for densest  $k$ -subgraph. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 201–210, 2010. doi:10.1145/1806689.1806719.
- 7 Aditya Bhaskara, Moses Charikar, Aravindan Vijayaraghavan, Venkatesan Guruswami, and Yuan Zhou. Polynomial integrality gaps for strong sdp relaxations of densest  $k$ -subgraph. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’12*, page 388–405, USA, 2012. Society for Industrial and Applied Mathematics.
- 8 Polina Bombina and Brendan Ames. Convex optimization for the densest subgraph and densest submatrix problems, 2019. arXiv:1904.03272.
- 9 Mark Braverman, Young Kun Ko, Aviad Rubinfeld, and Omri Weinstein. Eth hardness for densest- $k$ -subgraph with perfect completeness. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’17*, pages 1326–1341, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3039686.3039772>.
- 10 Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization, APPROX ’00*, pages 84–95, Berlin, Heidelberg, 2000. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=646688.702972>.
- 11 Amin Coja-Oghlan. Colouring semirandom graphs. *Comb. Probab. Comput.*, 16(4):515–552, July 2007. doi:10.1017/S0963548306007917.
- 12 Roe David and Uriel Feige. On the effect of randomness on planted 3-coloring models. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, STOC ’16*, pages 77–90, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897561.
- 13 Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *J. Comput. Syst. Sci.*, 63(4):639–671, December 2001. doi:10.1006/jcss.2001.1773.
- 14 Uriel Feige, Guy Kortsarz, and David Peleg. The dense  $k$ -subgraph problem. *Algorithmica*, 29(3):410–421, 2001. doi:10.1007/s004530010050.
- 15 Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001. doi:10.1006/jagm.2001.1183.
- 16 Uriel Feige and Michael Seltser. On the densest  $k$ -subgraph problem. *Algorithmica*, 29:2001, 1997.
- 17 G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, February 1989. doi:10.1137/0218003.
- 18 A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1984.

- 19 B. Hajek, Y. Wu, and J. Xu. Achieving exact cluster recovery threshold via semidefinite programming: Extensions. *IEEE Transactions on Information Theory*, 62(10):5918–5937, 2016.
- 20 Bruce Hajek, Yihong Wu, and Jiaming Xu. Computational Lower Bounds for Community Detection on Random Graphs. *arXiv e-prints*, page arXiv:1406.6625, June 2014. arXiv:1406.6625.
- 21 Bruce Hajek, Yihong Wu, and Jiaming Xu. Semidefinite programs for exact recovery of a hidden community. *Journal of Machine Learning Research*, 49(June):1051–1095, June 2016. 29th Conference on Learning Theory, COLT 2016 ; Conference date: 23-06-2016 Through 26-06-2016.
- 22 Ravi Kannan and V Vinay. *Analyzing the structure of large graphs*. Rheinische Friedrich-Wilhelms-Universität Bonn Bonn, 1999.
- 23 Subhash Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, December 2006. doi:10.1137/S0097539705447037.
- 24 Samir Khuller and Barna Saha. On finding dense subgraphs. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, ICALP '09*, pages 597–608, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-02927-1\_50.
- 25 Alexandra Kolla, Konstantin Makarychev, and Yury Makarychev. How to play unique games against a semi-random adversary: Study of semi-random models of unique games. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 443–452, 2011. doi:10.1109/FOCS.2011.78.
- 26 Anand Louis and Rakesh Venkat. Semi-random graphs with planted sparse vertex cuts: Algorithms for exact and approximate recovery. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 101:1–101:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.101.
- 27 Anand Louis and Rakesh Venkat. Planted models for k-way edge and vertex expansion. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, pages 23:1–23:15, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.23.
- 28 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Approximation algorithms for semi-random partitioning problems. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing, STOC '12*, pages 367–384, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214013.
- 29 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant factor approximation for balanced cut in the pie model. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC '14*, pages 41–49, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591841.
- 30 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Learning communities in the presence of errors. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1258–1291, Columbia University, New York, New York, USA, 2016. PMLR. URL: <http://proceedings.mlr.press/v49/makarychev16.html>.
- 31 Pasin Manurangsi. Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 954–961, 2017. doi:10.1145/3055399.3055412.
- 32 Theo McKenzie, Hermish Mehta, and Luca Trevisan. A new algorithm for the robust semi-random independent set problem. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 738–746, 2020. doi:10.1137/1.9781611975994.45.
- 33 F. McSherry. Spectral partitioning of random graphs. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 529–537, 2001.

- 34 Andrea Montanari. Finding one community in a sparse graph. *Journal of Statistical Physics*, 161, February 2015. doi:10.1007/s10955-015-1338-2.
- 35 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 755–764, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806792.
- 36 Prasad Raghavendra, David Steurer, and Prasad Tetali. Approximations for the isoperimetric and spectral profile of graphs and related parameters. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 631–640, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806776.
- 37 Anand Srivastav and Katja Wolf. Finding dense subgraphs with semidefinite programming. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '98, pages 181–191, London, UK, UK, 1998. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=646687.702946>.

# Sample-And-Gather: Fast Ruling Set Algorithms in the Low-Memory MPC Model

**Kishore Kothapalli**

IIIT Hyderabad, India

kkishore@iiit.ac.in

**Shreyas Pai** 

The University of Iowa, Iowa City, IA, USA

shreyas-pai@uiowa.edu

**Sriram V. Pemmaraju**

The University of Iowa, Iowa City, IA, USA

sriram-pemmaraju@uiowa.edu

---

## Abstract

---

Motivated by recent progress on symmetry breaking problems such as maximal independent set (MIS) and maximal matching in the low-memory Massively Parallel Computation (MPC) model (e.g., Behnezhad et al. PODC 2019; Ghaffari-Uitto SODA 2019), we investigate the complexity of ruling set problems in this model. The MPC model has become very popular as a model for large-scale distributed computing and it comes with the constraint that the memory-per-machine is strongly sublinear in the input size. For graph problems, extremely fast MPC algorithms have been designed assuming  $\tilde{\Omega}(n)$  memory-per-machine, where  $n$  is the number of nodes in the graph (e.g., the  $O(\log \log n)$  MIS algorithm of Ghaffari et al., PODC 2018). However, it has proven much more difficult to design fast MPC algorithms for graph problems in the *low-memory* MPC model, where the memory-per-machine is restricted to being strongly sublinear in the number of nodes, i.e.,  $O(n^\epsilon)$  for constant  $0 < \epsilon < 1$ .

In this paper, we present an algorithm for the 2-ruling set problem, running in  $\tilde{O}(\log^{1/6} \Delta)$  rounds whp, in the low-memory MPC model. Here  $\Delta$  is the maximum degree of the graph. We then extend this result to  $\beta$ -ruling sets for any integer  $\beta > 1$ . Specifically, we show that a  $\beta$ -ruling set can be computed in the low-memory MPC model with  $O(n^\epsilon)$  memory-per-machine in  $\tilde{O}(\beta \cdot \log^{1/(2^{\beta+1}-2)} \Delta)$  rounds, whp. From this it immediately follows that a  $\beta$ -ruling set for  $\beta = \Omega(\log \log \log \Delta)$ -ruling set can be computed in in just  $O(\beta \log \log n)$  rounds whp. The above results assume a total memory of  $\tilde{O}(m + n^{1+\epsilon})$ . We also present algorithms for  $\beta$ -ruling sets in the low-memory MPC model assuming that the total memory over all machines is restricted to  $\tilde{O}(m)$ . For  $\beta > 1$ , these algorithms are all substantially faster than the Ghaffari-Uitto  $\tilde{O}(\sqrt{\log \Delta})$ -round MIS algorithm in the low-memory MPC model.

All our results follow from a *Sample-and-Gather Simulation Theorem* that shows how random-sampling-based CONGEST algorithms can be efficiently simulated in the low-memory MPC model. We expect this simulation theorem to be of independent interest beyond the ruling set algorithms derived here.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Distributed algorithms

**Keywords and phrases** Massively Parallel Computation, Ruling Set, Simulation Theorems

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.28

**Related Version** A full version of this paper is available at: <https://arxiv.org/abs/2009.12477>.

**Funding** This work is funded in part by the Department of Science and Technology, Government of India, via Grant number MTR/2017/000849.



© Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 28; pp. 28:1–28:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

There has been considerable recent progress in the design and study of large-scale distributed computing models that are closer to reality, yet mathematically tractable. Of these, the *Massively Parallel Computing (MPC)* model [24, 37] has gained significant attention due to its flexibility and its ability to closely model existing distributed computing frameworks used in practice such as MapReduce [14], Spark [38], Pregel [32], and Giraph [11].

The MPC model is defined by a set of machines, each having at most  $S$  words of memory. The machines are connected to each other via an all-to-all communication network. Communication and computation in this model are synchronous. In each round, each machine receives up to  $S$  words from other machines, performs local computation, and sends up to  $S$  words to other machines. The key characteristic of the MPC model is that both the memory upper bound  $S$  and the number of machines used are assumed to be strongly sublinear in the input size  $N$ , i.e., bounded by  $O(N^{1-\varepsilon})$  for some constant  $\varepsilon$ ,  $0 < \varepsilon < 1$ . This characteristic models the fact that in modern large-scale computational problems the input is too large to fit in a single machine and is much larger than the number of available machines.

Even though the MPC model is relatively new, a wide variety of classical graph problems have been studied in this model. This stream of research includes the design of fast algorithms [4, 6, 13, 12, 21] as well as lower bound constructions [10, 20, 35]. A particular, though not exclusive, focus of this research has been on *symmetry breaking* problems such as maximal independent set (MIS) [6, 21, 18], maximal matching [7], and  $(\Delta + 1)$ -coloring [9, 3], along with related graph optimization problems such as minimum vertex cover and maximum matching.

For graph problems, the input size is  $\tilde{O}(m + n)$  where  $m$  is the number of edges and  $n$  is the number of nodes of the input graph. Thus,  $O((m + n)^{1-\varepsilon})$ , for some constant  $\varepsilon$ ,  $0 < \varepsilon < 1$ , is an upper bound on both the number of machines that can be used and the size  $S$  of memory per machine. It turns out that the difficulty of graph problems varies significantly based on how  $S$  relates to the number of nodes ( $n$ ) of the input graph. Specifically, three regimes for  $S$  have been considered in the literature.

- **Strongly superlinear memory** ( $S = O(n^{1+\varepsilon})$ ): For this regime to make sense in the MPC model, the input graph needs to be highly dense, i.e.,  $m \gg S \gg n$  such that  $S$  is strongly sublinear in  $m$ . Even though the input graph is dense, the fact that each machine has  $O(n^{1+\varepsilon})$  local memory makes this model quite powerful. For example, in this model, problems such as minimum spanning tree, MIS, and 2-approximate minimum vertex cover, all have  $O(1)$ -round algorithms [24, 22].
- **Near-linear memory** ( $S = \tilde{O}(n)$ ): Problems become harder in this regime, but symmetry breaking problems such as MIS, approximate minimum vertex cover, and maximal matching can still be solved in  $O(\log \log n)$  rounds [13, 2, 17, 19]. Furthermore, recently Assadi, Chen, and Khanna [3] presented an  $O(1)$ -round algorithm for  $(\Delta + 1)$ -vertex coloring.
- **Strongly sublinear memory** ( $S = O(n^\varepsilon)$ ): Problems seem to get much harder in this regime and whether there are sublogarithmic-round algorithms for certain graph problems in this regime is an important research direction. For example, it is conjectured that the problem of distinguishing if the input graph is a single cycle vs two disjoint cycles of length  $n/2$  requires  $\Omega(\log n)$  rounds [37, 20]. However, even in this regime, Ghaffari and Uitto [21] have recently shown that MIS does have a sublogarithmic-round algorithm, running in  $\tilde{O}(\sqrt{\log \Delta})$  rounds, where  $\Delta$  is the maximum degree of the input graph. This particular result serves as a launching point for the results in this paper.

The MIS problem has been called “a central problem in the area of locality in distributed computing” (2016 Dijkstra award citation). Starting with the elegant, randomized MIS algorithms from the mid-1980s by Luby [31] and by Alon et al. [1], several decades of research has now been devoted to designing MIS algorithms in various models of parallel and distributed computing (e.g., PRAM, LOCAL, CONGEST, CONGESTED-CLIQUE, and MPC). A *ruling set* is a natural relaxation of an MIS and considerable research has been devoted to solving the ruling set problem in different models of distributed computation as well [5, 26, 8, 15]. An  $(\alpha, \beta)$ -*ruling set* of a graph  $G = (V, E)$  is a subset  $S \subseteq V$  such that (i) every pair of nodes in  $S$  are at distance at least  $\alpha$  from each other and (ii) every node in  $V$  is at distance at most  $\beta$  from some node in  $S$ . An MIS is just a  $(2, 1)$ -ruling set. Research on the ruling set problem has focused on the question of how much faster distributed ruling set algorithms can be relative to MIS algorithms and whether there is a provable separation in the distributed complexity of these problems in different models of distributed computing. For example, in the LOCAL model<sup>1</sup>, Kuhn, Moscibroda, and Wattenhofer [27, 28] show an  $\Omega\left(\min\left\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{\log n}{\log \log n}}\right\}\right)$  lower bound for MIS, even for randomized algorithms. However, combining the recursive sparsification procedure of Bisht et al. [8] with the improved MIS algorithm of Ghaffari [15] and the recent deterministic network decomposition algorithm of Rozhon and Ghaffari [36], it is possible to compute  $\beta$ -ruling sets in  $O(\beta \log^{1/\beta} \Delta + \text{polyloglog}(n))$  rounds, thus establishing a separation between these problems, even for  $\beta = 2$ , in the LOCAL model. In this paper, we are interested only in  $(2, \beta)$ -ruling sets and so as a short hand, we drop the first parameter “2” and call these objects  $\beta$ -ruling sets. Also as a short hand, we will use *low-memory MPC model* to refer to the *strongly sublinear memory MPC model*. As mentioned earlier, Ghaffari and Uitto [21] recently presented an algorithm that solves MIS in the low-memory MPC model in  $\tilde{O}(\sqrt{\log \Delta})$  rounds. However, nothing more is known about the 2-ruling set problem in this model and the fastest 2-ruling set algorithm in the low-memory MPC model is just the above-mentioned MIS algorithm. This is in contrast to the situation in the linear-memory MPC model. In this model, the fastest algorithm for solving MIS runs in  $O(\log \log n)$  rounds [17], whereas the fastest 2-ruling set algorithm runs in  $O(\log \log \log n)$  rounds [23]. This distinction between the status of MIS and 2-ruling sets in the linear-memory MPC model prompts the following related questions.

*Is it possible to design an  $o(\sqrt{\log \Delta})$ -round, 2-ruling set algorithm in the low-memory MPC model? Could we in fact design 2-ruling set algorithms in the low-memory MPC model that run in  $O(\text{polyloglog}(n))$  rounds?*

## 1.1 Main Results

We make progress on the above question via the following results proved in this paper.

1. We show (in Theorem 19 part (i)) that a 2-ruling set of a graph  $G$  can be computed in  $\tilde{O}(\log^{1/6} \Delta)$  rounds in the low-memory MPC model. We generalize this result to  $\beta$ -ruling sets, for  $\beta \geq 2$  (in Theorem 23 part (i)), and show that a  $\beta$ -ruling set of a graph  $G$  can be computed in  $\tilde{O}(\log^{1/(2^{\beta+1}-2)} \Delta)$  rounds in the low-memory MPC model. These algorithms are substantially faster than the MIS algorithm [21] for the low-memory MPC model. The inverse exponential dependency on  $\beta$  in the running time of the  $\beta$ -ruling set algorithm is

<sup>1</sup> The LOCAL model is a synchronous, message passing model of distributed computation [29, 34] with unbounded messages. See Section 1.2 for definitions of related models of computation.

worth noting. This dependency implies that for any  $\beta = \Omega(\log \log \log \Delta)$ , we can compute a  $\beta$ -ruling set in only  $O(\beta \cdot \text{polyloglog}(n))$  rounds. This is in contrast to the situation in the LOCAL model; using the  $O(\beta \cdot \log^{1/\beta} \Delta + \text{polyloglog}(n))$ -round  $\beta$ -ruling set algorithm in the LOCAL model mentioned earlier, one can obtain an  $O(\text{polyloglog}(n))$ -round algorithm only for  $\beta = \Omega(\log \log \Delta)$ .

2. Even though the above-mentioned results are in the low-memory MPC model, they assume no restrictions on the total memory used by all the machines put together. Specifically, we obtain the above results allowing a total of  $\tilde{O}(m + n^{1+\epsilon})$  memory. Note that the input uses  $\tilde{O}(m)$  memory and thus these algorithms make use of  $\tilde{O}(n^{1+\epsilon})$  extra total memory. If we place the restriction that the total memory cannot exceed the input size, i.e.,  $\tilde{O}(m)$ , then we get slightly weaker results. Specifically, we show (in Theorem 19 part (ii)) that a 2-ruling set can be computed in  $\tilde{O}(\log^{1/4} \Delta)$  rounds in the low-memory MPC model using  $\tilde{O}(m)$  total memory. Additionally, we show (in Theorem 23 part (ii)) that a  $\beta$ -ruling set, for any  $\beta \geq 1$ , can be computed in  $\tilde{O}(\log^{1/2\beta} \Delta)$  rounds in the low-memory MPC model using  $\tilde{O}(m)$  total memory. Note that even though these results are weaker than those we obtain in the setting where total memory is unrestricted, for  $\beta > 1$ , these algorithms are much faster than the  $\tilde{O}(\sqrt{\log \Delta})$ -round, low-memory MPC model algorithm for MIS by Ghaffari and Uitto [21] that uses  $\tilde{O}(m)$  total memory. Also note that by plugging in  $\beta = 1$ , we recover the Ghaffari-Uitto MIS algorithm.

**Technical Contributions.** We obtain all of these results by applying new Simulation Theorems (Theorems 9 and 12) that we develop and prove. These Simulation Theorems provide a general method for deriving fast MPC algorithms from known distributed algorithms in the CONGEST model<sup>2</sup> and they form the main technical contribution of this paper.

A well-known technique [16, 21, 23, 33] for designing fast algorithms in “all-to-all” communication models such as MPC is the following “ball doubling” technique. Informally speaking, if every node  $v$  knows the state of the  $k$ -neighborhood around  $v$ , then by exchanging this information with all nodes, ideally in  $O(1)$  rounds, it is possible to learn the state of the  $2k$ -neighborhood around each node. In this manner, nodes can learn the state their  $\ell$ -neighborhood in  $O(\log \ell)$  rounds. Then it is possible to simply use local computation at each node to “fast forward” the algorithm by  $\ell$  rounds, without any further communication. In this manner, a phase consisting of  $\ell$  rounds in the CONGEST model can be compressed into  $O(\log \ell)$  rounds in the MPC model. This description of the “ball doubling” technique completely ignores the main obstacle to using this technique: the  $k$ -neighborhoods around nodes may be so large that bandwidth constraints of the communication network may disallow rapid exchange of these  $k$ -neighborhoods.

Our main contribution is to note that in many randomized, distributed algorithms in the CONGEST model, there is a natural *sparsification* that occurs, i.e., in each round a randomly sampled subset of the nodes are active, and the rest are silent. This implies that the  $k$ -neighborhoods that are exchanged only need to involve sparse subgraphs induced by the sampled nodes. A technical challenge we need to overcome is that the subgraph induced by sampled nodes is not just from the next round, but from the  $\ell$  future rounds; so we need to be able to estimate which nodes will be sampled in the future. On the basis of this idea, we introduce the notion of  $\alpha$ -*sparsity* of a randomized CONGEST algorithm, for a parameter  $\alpha$ ; basically smaller the  $\alpha$  greater the sparsification induced by random sampling. We present

---

<sup>2</sup> The CONGEST model [34] is similar to the LOCAL model except that in the CONGEST model there is an  $O(\log n)$  bound on the size of each message.

*Sample-and-Gather* Simulation Theorems in which, roughly speaking, an  $R$ -round CONGEST algorithm is simulated in  $\tilde{O}(R/\sqrt{\log_\alpha n})$  rounds (respectively,  $\tilde{O}(R/\sqrt{\log_\alpha \Delta})$  rounds) in the low-memory MPC model, where the total memory is  $\tilde{O}(m + n^{1+\varepsilon})$  (respectively,  $\tilde{O}(m)$ ).

Our Simulations Theorems are inspired by a Simulation Theorem due to Behnezhad et al. [6, Lemma 5.5]. Using their Simulation Theorem, an  $R$ -round state-congested algorithm can be simulated in (roughly)  $R/\log_\Delta n$  low-memory MPC rounds. In contrast, our Simulation Theorem (Theorem 9) yields a running time of (roughly)  $R/\sqrt{\log_\alpha n}$ , where  $\alpha$  is a sparsity parameter. When the input graph has high maximum degree, but the state-congested algorithm samples a very sparse subgraph (i.e.,  $\alpha$  is small) then our Simulation Theorems provide a huge advantage over the Behnezhad et al. Simulation Theorems.

To obtain our results for ruling sets, we apply the Sample-and-Gather Simulation Theorems to the sparsification procedure of Kothapalli and Pemmaraju [26] and Bisht et al. [8] and to the sparsified MIS algorithm of Ghaffari [16]. We note that by applying the Sample-and-Gather Simulation Theorems to the sparsified MIS algorithm of Ghaffari [16], we recover the Ghaffari-Uitto low-memory MPC algorithm for MIS [21], built from scratch. We believe that the Sample-and-Gather Theorems will be of independent interest because they simplify the design of fast MPC algorithms.

## 1.2 Technical Preliminaries

**Notation.** For a node  $v \in V$  we denote its non-inclusive neighborhood in  $G$  by  $\text{Nbr}(v)$ . Moreover, we define  $\text{Nbr}^+(v) = \text{Nbr}(v) \cup \{v\}$ ,  $\text{Nbr}(S) = \bigcup_{v \in S} \text{Nbr}(v)$ ,  $\text{Nbr}^+(S) = \bigcup_{v \in S} \text{Nbr}^+(v)$ . The standard usage of the  $\tilde{O}(f(n))$  notation is to denote  $O(\text{poly log}(f(n)) \cdot f(n))$ . But, because our round and memory complexity bounds involve multiple parameters (e.g.,  $n$ ,  $m$ , and  $\Delta$ ), we abuse notation and use the  $\tilde{O}(\cdot)$  notation to hide  $\text{poly log } n$  or  $\text{poly log log } n$  factors, as appropriate (e.g.,  $\tilde{O}(\log^{1/6} \Delta)$  denotes  $O(\log^{1/6} \Delta \cdot \text{poly log log } n)$ ). Moreover, we consider  $\varepsilon$  to be a constant in  $(0, 1)$  and hence, we don't explicit mention  $\varepsilon$  dependency in the run time results. However the dependency on epsilon is of the form  $1/\varepsilon^c$  for some small constant  $c \geq 1$  (and not  $2^{-\varepsilon^c}$ ).

**Distributed Computing Models.** In the CONGEST model [34] a communication network is abstracted as an  $n$ -node graph. In synchronous rounds each node can send a  $O(\log n)$  bit message to each of its neighbors. The CONGESTED-CLIQUE model is similar to the CONGEST model, but nodes can send  $O(\log n)$ -bits messages to all other nodes, not only to its neighbors in the input graph  $G$  [30]. The LOCAL model [29] is the same as the CONGEST model, except the message sizes can be unbounded.

**MPC model.** Typically, in the MPC model, it is assumed that the input graph is distributed in a node-centric manner among the machines. In other words, for each node  $v$ , there is a machine  $M_v$  that hosts it and  $M_v$  knows all the neighbors of  $v$  and the machines that host these neighbors. However, this scheme cannot be implemented as-is in the low-memory MPC model because the degree of a node could be larger than the memory volume  $n^\varepsilon$  of a machine. To deal with this issue, we first assume that a node  $v$  with  $\text{deg}(v) > n^\varepsilon$  is split into copies that are distributed among different machines and we have a virtual  $O(1/\varepsilon)$ -depth balanced tree on these copies of  $v$ . The root of this tree coordinates communication between  $v$  and its neighbors in the input graph. By itself, this is insufficient because information from  $v$ 's neighbors cannot travel up  $v$ 's tree without running into a memory bottleneck. However, if computation at each node can be described by a *separable* function, then this is possible.

The following definition of separable functions captures functions such as max, min, sum, etc. This issue and the proposed solution have been discussed in [21, 6].

► **Definition 1.** Let  $f : 2^{\mathbb{R}} \rightarrow \mathbb{R}$  denote a set function. We call  $f$  separable iff for any set of reals  $A$  and for any  $B \subseteq A$ , we have  $f(A) = f(f(B), f(A \setminus B))$ .

The following lemma [6] shows that it is possible to compute the value of a separable function  $f$  on each of the nodes in merely  $O(1/\varepsilon)$  rounds. The bigger implication of this lemma is that a single round of a CONGEST algorithm can be simulated in  $O(1/\varepsilon)$  low-memory MPC rounds.

► **Lemma 2** ([6]). Suppose that on each node  $v \in V$ , we have a number  $x_v$  of size  $O(\log n)$  bits and let  $f$  be a separable function. There exists an algorithm that in  $O(1/\varepsilon)$  rounds of MPC, for every node  $v$ , computes  $f(\{x_u \mid u \in \text{Nbr}(v)\})$  whp in the low-memory MPC model with  $\tilde{O}(n)$  total memory.

**Note about proofs.** Due to space constraints, we only include two proofs of the main Sample-and-Gather Simulation Theorem in the paper; a full version of the paper, with all proofs, is available at <https://arxiv.org/abs/2009.12477>, [25].

## 2 The Sample-and-Gather Simulation

Our simulation theorems apply to a subclass of CONGEST model algorithms called *state-congested* algorithms [6].

- **Definition 3.** An algorithm in the CONGEST model is said to be *state-congested* if
- (i) by the end of round  $r$ , for any  $r$ , at each node  $v$ , the algorithm stores a state  $\sigma_r(v)$  of size  $O(\deg(v) \text{polylog}(n))$  bits, i.e., an average of  $O(\text{polylog}(n))$  bits per neighbor. The initial state  $\sigma_0(v)$  of each node  $v$  is its ID. Furthermore, the computation performed by each node  $v$  in each round  $r$  uses an additional temporary space of size  $O(\deg(v) \cdot \text{polylog}(n))$  bits.
  - (ii) The states of the nodes after the last round of the algorithm are sufficient in determining, collectively, the output of the algorithm.

A key feature of a state-congested algorithm is that the local state at each node stays bounded in size throughout the execution of the algorithm.

We inductively design a fast low-memory MPC algorithm that simulates a given state-congested algorithm. For this purpose, we start by assuming that we have a state-congested, possibly randomized, algorithm  $Alg$ , whose first  $t$  rounds have been correctly simulated in the low-memory MPC model. Our goal now is to simulate a *phase* consisting of the next  $\ell$  rounds of  $Alg$ , i.e., rounds  $t+1, t+2, \dots, t+\ell$ , in just  $O(\log \ell)$  low-memory MPC rounds. We categorize each node  $u$  in a round  $\tau$ ,  $t+1 \leq \tau \leq t+\ell$ , based on its activity in round  $\tau$ . Specifically, a node  $u$  is a *sending node* in round  $\tau$  if sends at least one message in round  $\tau$ . Moreover, a node is called a *oblivious node* if it does not update its state in round  $\tau$ . In other words, an oblivious node ignores any messages it receives in round  $\tau$ .

Consider a node  $u$  at the start of the phase we want to compress. Since this is immediately after round  $t$ , node  $u$  knows its local state  $\sigma_t(u)$ . Let  $p_{t+1}(u)$  denote the probability that node  $u$  is a sending node in round  $t+1$ . We call this the *activation probability* of node  $u$  in round  $t+1$ . Also, for any node  $v$ , let  $A_{t+1}(v) := \sum_{u \in \text{Nbr}(v)} p_{t+1}(u)$  denote the *activity level* in  $v$ 's neighborhood in round  $t+1$ . Note that  $p_{t+1}(u)$  is completely determined by  $\sigma_t(u)$  and so node  $u$  can locally calculate  $p_{t+1}(u)$  after round  $t$ . In order to simulate rounds

$t + 1, t + 2, \dots, t + \ell$  in a compressed fashion in the MPC model, every node  $u$  needs to know the probability of it being a sending node in each of these rounds. But, rounds  $t + 2, t + 3, \dots, t + \ell$  are in the future and so node  $u$ , using current knowledge, can only *estimate* an upper bound  $\tilde{p}_\tau(u)$  on the probability that it will be a sending node in round  $\tau$ ,  $t + 2 \leq \tau \leq t + \ell$ .

In general, doing this estimation can be difficult because sampling probabilities of a node  $u$  in the  $\ell$  future rounds depend on current states of nodes in an  $\ell$ -radius neighborhood around node  $u$ . The volume of such a neighborhood may be too high to fit in the memory of any machine in the low-memory MPC model. In fact, our Sample-and-Gather Simulation Theorems are designed to avoid exactly this type of ball gathering of potentially dense neighborhoods. It turns out that this estimation is essentially trivial for the two applications of our Simulation Theorem described in Section 3.1. This is because for these algorithms, sampling probabilities for all active nodes increase by a known multiplicative factor in each round. Thus independent of a node  $u$ 's future state (e.g., whether it is active), it is possible to obtain an upper bound, denoted  $\tilde{p}_\tau(u)$ , that node  $u$  will be a sending node in round  $\tau$ , for rounds  $\tau = t + 2, t + 3, \dots, t + \ell$ . For round  $\tau = t + 1$ , we simply set  $\tilde{p}_{t+1}(u) := p_{t+1}(u)$ , i.e., the estimated activation probability in round  $t + 1$  is the actual activation probability.

For any  $\tau$ ,  $t + 1 \leq \tau \leq t + \ell$ , for any node  $v$ , let  $\tilde{A}_\tau(v) := \sum_{u \in N_{br}(v)} \tilde{p}_\tau(u)$  denote the *estimated activity level* in node  $v$ 's neighborhood in round  $\tau$ . Note that for the first round in the phase,  $\tau = t + 1$ , the estimated and actual activity levels are identical. Finally, let  $\tilde{A}_\tau$  be the maximum  $\tilde{A}_\tau(v)$ , where the maximum is taken over all nodes  $v$  that are not oblivious nodes. The maximum being taken over all non-oblivious nodes is motivated by the fact that if a node is oblivious, it does not update its state and therefore the activity level in its neighborhood is not relevant.

► **Lemma 4.** *Suppose  $\ell$  is such that*

$$\left( \sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n \right)^\ell \leq O(n^{\varepsilon/2}). \quad (1)$$

*Then the next phase of the algorithm Alg consisting of rounds  $t + 1, t + 2, \dots, t + \ell$  can be simulated in  $O(\log \ell)$  rounds in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory.*

**Proof.** Simulating rounds  $t + 1, t + 2, \dots, t + \ell$  of algorithm Alg is equivalent to computing the state  $\sigma_{t+\ell}(v)$  for every node  $v \in V$ . We use the 2-step algorithm below to do this computation. First, we introduce some notation. Let  $B_G(v, \ell)$  denote the labeled subgraph of  $G$ , induced by nodes that are at most  $\ell$  hops from  $v$  in  $G$  and in which each node  $u$  is labeled with its local state  $\sigma_t(u)$  after round  $t$ .

**Step 1:** For each node  $v \in V$ , designate a distinct machine  $M_v$  at which we gather a “sampled” subgraph  $S_G(v, \ell)$  of  $B_G(v, \ell)$ . The definition of  $S_G(v, \ell)$  is provided below.

**Step 2:** Using the subgraph  $S_G(v, \ell)$ , machine  $M_v$  locally simulates rounds  $t + 1, t + 2, \dots, t + \ell$  of Alg and computes  $\sigma_{t+\ell}(v)$ .

In the rest of the proof, we will first define the subgraph  $S_G(v, \ell)$ . We will then show in Claim 5 that using this subgraph, it is possible for machine  $M_v$  to locally simulate rounds  $t + 1, t + 2, \dots, t + \ell$  of Alg. We then show in Claim 6 that assuming  $\ell$  satisfies (1), the size of  $S_G(v, \ell)$  is  $O(n^\varepsilon)$  whp. Finally, in Claim 7, we show that the subgraph  $S_G(v, \ell)$  can be gathered at each machine  $M_v$  in parallel in  $O(\log \ell)$  rounds. These claims together complete the proof of the lemma.

Each node  $u \in V$  generates a sequence of uniformly distributed random bits  $r_\tau^1(u)$ ,  $r_\tau^2(u)$ ,  $\dots$ ,  $r_\tau^{c \cdot \log n}(u)$  for a large enough constant  $c$ . These bits are designated for round  $\tau$ ,  $t+1 \leq \tau \leq t+\ell$  and they serve two purposes: (i) they are used to randomly sample  $u$  based on the estimate  $\tilde{p}_\tau(u)$  that  $u$  will be a sending node in round  $\tau$ , and (ii) they are used to simulate  $u$ 's actions in round  $\tau$ . It is important that the same bits be used for both purposes so that there is consistency in  $u$ 's random actions. Specifically,  $u$  constructs a real number  $R_\tau(u)$  that is uniformly distributed over  $\{i/2^{c \cdot \log n} \mid 0 \leq i < c \log n\}$  using these bits. Node  $u$  adds these  $O(\ell \cdot \log n)$  bits to its local state after round  $t$ ,  $\sigma_t(u)$ . Node  $u$  then *marks itself for round  $\tau$*  if  $R_\tau(u) \leq p_\tau(u)$ . If a node  $u$  marks itself for a round  $\tau$  it means that in  $u$ 's estimate after round  $t$ ,  $u$  will be a sending node in round  $\tau$ . Further, node  $u$  is *marked* if it is marked for round  $\tau$  for any  $\tau$ ,  $t+1 \leq \tau \leq t+\ell$ . The ‘‘sampled’’ subgraph  $S_G(v, \ell)$  is the subgraph of  $B_G(v, \ell)$  induced by  $v$  along with all nodes  $u$  in  $B_G(v, \ell)$  that are marked.

▷ **Claim 5.** For any node  $v \in V$ , information in  $S_G(v, \ell)$  is enough to locally compute  $\sigma_{t+\ell}(v)$ .

*Proof.* We prove this claim inductively. Specifically, we prove the following:

For any  $i$ ,  $0 < i \leq \ell$ , in addition to knowing  $S_G(v, \ell)$ , if we know the states  $\sigma_{t+\ell-i}(u)$  for all  $u \in S_G(v, i)$  then we can compute the states  $\sigma_{t+\ell-i+1}(u)$  for all  $u \in S_G(v, i-1)$ .

The premise of this statement is true for  $i = \ell$  because  $S_G(v, \ell)$  contains the round- $t$  local states  $\sigma_t(u)$  for all  $u \in S_G(v, \ell)$ . For  $i = 1$  this claim is equivalent to saying that in addition to  $S_G(v, \ell)$ , if we know  $\sigma_{t+\ell-1}(u)$  for all neighbors of  $v$  in  $S_G(v, \ell)$  then we can compute  $\sigma_{t+\ell}(v)$ . This is what we need to show.

To be able to compute  $\sigma_{t+\ell-i+1}(u)$  for any  $u$  in  $S_G(v, i-1)$ , we need to know the round- $(t+\ell-i)$  local states  $\sigma_{t+\ell-i}(w)$  for all neighbors  $w$  of  $u$  that are sending nodes in round  $t+\ell-i$ . With high probability, the probability  $p_w$  that a neighbor  $w$  of  $u$  sends messages in round  $t+\ell-i$  is upper bounded by the estimate  $\tilde{p}_{t+\ell-i}(w)$  that  $w$  computed after round  $t$ . Node  $w$  sends messages in round  $t+\ell-i$  if  $R_{t+\ell-i}(w) \leq p_w$ . Since  $p_w \leq \tilde{p}_{t+\ell-i}(w)$ , we know that  $R_{t+\ell-i}(w) \leq \tilde{p}_{t+\ell-i}(w)$  and therefore  $w$  is marked and included in  $S_G(v, \ell)$ . Also note that since  $u \in S_G(v, i-1)$  and  $w$  is a neighbor of  $u$ , we see that  $w \in S_G(v, i)$ . Thus any node  $w$  that sends a message to node  $u$  in round  $t+\ell-i$  belongs to  $S_G(v, i)$  and by the hypothesis of the inductive claim we know  $\sigma_{t+\ell-i}(w)$ . With the knowledge of  $\sigma_{t+\ell-i}(w)$ , we can simulate round  $t+\ell-i+1$  at each node  $w$ , using the random real  $R_{t+\ell-i+1}(w)$  to execute any random actions  $w$  may take. Then using the message received by  $u$  from all such neighbors  $w$  in round  $t+\ell-i+1$ , we can update  $u$ 's local state, thus computing  $\sigma_{t+\ell-i+1}(u)$ . ◁

▷ **Claim 6.** For any node  $v \in V$ , the size of  $S_G(v, \ell)$  is at most  $\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell$  whp.

*Proof.* Consider an arbitrary  $v \in V$  and  $u \in B_G(v, \ell)$  and a round  $t+1 \leq \tau \leq t+\ell$ . Node  $u$  is marked for round  $\tau$  with probability  $p_\tau(u)$ . Recalling that  $Nbr(u)$  denotes the set of neighbors of  $u$  in  $G$ , we see that expected number of neighbors of  $u$  marked for round  $t+1 \leq \tau \leq t+\ell$  is at most

$$\sum_{w \in Nbr(u)} p_\tau(w) \leq \tilde{A}_\tau(u) \leq \tilde{A}_\tau.$$

Furthermore, since neighbors of  $u$  are marked for round  $\tau$  independently, by Chernoff bounds we see that the number of neighbors that  $u$  has in  $S_G(v, \ell)$  that are marked for round  $\tau$  is  $\tilde{A}_\tau \log n$  whp. By the union bound this means that the number of neighbors that  $u$  has in  $S_G(v, \ell)$  is  $\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n$  whp. From this it follows that the size of  $S_G(v, \ell)$  is  $\left(\sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n\right)^\ell$ . ◁

▷ **Claim 7.** For every node  $v \in V$ , the graph  $S_G(v, \ell)$  can be gathered at  $M_v$  in at most  $O(\log \ell)$  rounds.

*Proof.* Here we use the “ball doubling” technique that appears in a number of papers on algorithms in “all-to-all” communication models (e.g., [16, 21, 23, 33]). Suppose that each machine  $M_v$  knows  $S_G(v, i)$  for some  $0 \leq i \leq \ell/2$ . Each machine  $M_v$  then sends  $S_G(v, i)$  to machine  $M_u$  for every node  $u$  in  $S_G(v, i)$ . After this communication is completed, each machine  $M_v$  can construct  $S_G(v, 2i)$  from the information it has received because  $S_G(v, 2i)$  is contained in the union of  $S_G(u, i)$  for all  $u$  in  $S_G(v, i)$ .

We now argue that this communication can be performed in  $O(1)$  rounds. First, note that the size of  $S_G(v, i)$  is bounded above by  $O(n^{\varepsilon/2})$ . This also means that  $S_G(v, i)$  contains  $O(n^{\varepsilon/2})$  nodes. Therefore,  $M_v$  needs to send a total of  $O(n^{\varepsilon/2}) \times O(n^{\varepsilon/2}) = O(n^\varepsilon)$  words. A symmetric argument shows an  $O(n^\varepsilon)$  bound on the number of words  $M_v$  receives. Since  $O(n^\varepsilon)$  words can be sent and received in each communication round, this communication can be completed in  $O(1)$  rounds. ◁

With the claims proven, we finish the proof of the Lemma. ◀

The biggest benefit from using this “sample-and-gather” simulation approach is for state-congested algorithms that sample a sparse subgraph and all activity occurs on this subgraph. We formalize this sparse sampling property as follows.

► **Definition 8.** Consider a state-congested algorithm  $Alg$  that completes in  $R$  rounds. For a parameter  $\alpha \geq 2$ , we say that  $Alg$  is  $\alpha$ -sparse if for all positive integers,  $t$  and  $\ell$  satisfying  $t + \ell \leq R$ , for a length- $\ell$  phase of  $Alg$  starting at round  $t + 1$  the following two properties hold.

(a) **Bounded activity level:** The activity level in the first round of the phase,  $A_{t+1}$ , satisfies the property:  $A_{t+1} = O(\alpha^\ell \cdot \log n)$ .

(b) **Bounded growth of estimated activity level:** The estimated activity level  $\tilde{A}_\tau$ ,  $t+1 \leq \tau \leq t+\ell$ , shows bounded growth. Specifically,  $\tilde{A}_{\tau+1} \leq \alpha \tilde{A}_\tau$  for all  $t+1 \leq \tau \leq t+\ell-1$ .

Together these properties require the activity level in each neighborhood to be low (Property (a)), but also that the *estimated* activity level of each node does not grow too fast in future rounds (Property (b)). When these two properties hold, Lemma 4 can be applied inductively to obtain the following theorem. The fact that we use a single parameter  $\alpha$  as an upper bound for both Properties (a) and (b) is just a matter of convenience and leads to an easy-to-state bound on number of rounds in this theorem.

► **Theorem 9 (Sample-and-Gather Theorem v1).** Let  $Alg$  be an  $\alpha$ -sparse state-congested algorithm that completes in  $R$  rounds. Then  $Alg$  can be simulated in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory, for constant  $0 < \varepsilon < 1$ , in  $O(R \log \log n / \sqrt{\log_\alpha n})$  rounds.

**Proof.** Let  $\ell = \lfloor \sqrt{\frac{\varepsilon}{8} \cdot \log_\alpha n} \rfloor$ . Partition the  $R$  rounds of  $Alg$  into  $\lceil R/\ell \rceil$  phases, where Phase  $i$ ,  $1 \leq i < \lceil R/\ell \rceil$ , consists of the  $\ell$  rounds  $(i-1) \cdot \ell + 1, (i-1) \cdot \ell + 2, \dots, i \cdot \ell$  and Phase  $\lceil R/\ell \rceil$  consists of at most  $\ell$  rounds  $(\lceil R/\ell \rceil - 1) \cdot \ell + 1, (\lceil R/\ell \rceil - 1) \cdot \ell + 2, \dots, R$ .

We now use the fact that  $Alg$  is  $\alpha$ -sparse to show, via series of inequalities, that  $\ell$  satisfies Inequality (1).

$$\begin{aligned}
 \left( \sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \cdot \log n \right)^\ell &\leq \left( \tilde{A}_{t+1} \cdot \log n \cdot \sum_{i=0}^{\ell-1} \alpha^i \right)^\ell && \text{(by Property (b) of being } \alpha\text{-sparse)} \\
 &\leq \left( A_{t+1} \cdot \log n \cdot \sum_{i=0}^{\ell-1} \alpha^i \right)^\ell && \text{(by } \tilde{A}_{t+1} = A_{t+1}\text{)} \\
 &\leq \left( \alpha^\ell \cdot \log^2 n \cdot \sum_{i=0}^{\ell-1} \alpha^i \right)^\ell && \text{(by Property (a) of being } \alpha\text{-sparse)} \\
 &= \left( \alpha^\ell \cdot \log^2 n \cdot \frac{\alpha^\ell - 1}{\alpha - 1} \right)^\ell && \text{(by geometric series)} \\
 &\leq \alpha^{2\ell^2} \cdot (\log^2 n)^\ell && \text{(by } \ell \geq 1, \alpha \geq 2\text{)} \\
 &\leq n^{\varepsilon/4} \cdot n^{o(1)} && \text{(by } \ell = \lfloor \sqrt{\frac{\varepsilon}{8} \cdot \log_\alpha n} \rfloor \text{)} \\
 &\leq n^{\varepsilon/2}.
 \end{aligned}$$

By using Lemma 4, this implies that each phase can be simulated in the MPC models with  $O(n^\varepsilon)$  memory per machine in  $O(\log \ell) = O(\log \log n)$  rounds. Given that the  $R$  rounds of  $Alg$  are partitioned into  $\lceil R/\ell \rceil$  phases, we see that  $Alg$  can be implemented in the MPC model with  $O(n^\varepsilon)$  memory per machine in  $O(R \log \log n / \sqrt{\varepsilon \log_\alpha n})$  rounds. ◀

Theorem 9 provides a Simulation Theorem for the MPC model in which machines use  $O(n^\varepsilon)$  memory per machine. However, the total memory used by MPC algorithms that result from this theorem is  $\tilde{O}(m + n^{1+\varepsilon})$ . We now show that under fairly general circumstances, it is possible to obtain a Simulation Theorem yielding low-memory MPC algorithms that use only  $\tilde{O}(m)$  total memory, while taking slightly more time.

► **Definition 10.** A CONGEST algorithm  $Alg$  is said to be degree-ordered if it satisfies two properties.

- (a) The execution of  $Alg$  can be partitioned into Stages  $1, 2, \dots$  such that in Stage  $i$  the only active nodes are those whose degree is greater than  $\Delta^{1/2^i}$  and other nodes that are neighbors of these “high degree” nodes.
- (b) Let  $R_i$  be the number of rounds in Stage  $i$ . Then  $R_i \leq R_{i-1}/2$ .

A lot of symmetry breaking algorithms are either inherently degree-ordered or can be made so with small modifications – this can be seen in the applications of the Sample-and-Gather Theorems in Section 3.1. The fact that our definition permits activity in a stage not just at nodes that are “high degree” for that stage, but even at other nodes that are neighbors of high degree nodes, provides the flexibility we need for our applications. In fact, it is possible to further relax this definition and allow all nodes within  $O(1)$  hops of “high degree” nodes to be active in a stage; for ease of exposition we just work with the current definition. For algorithms that are degree-ordered, we can gather balls centered at active nodes, whose volume is at most the degree threshold for the current stage. This allows us to use a simple charging scheme to charge the sizes of the balls to the memory already allocated for the neighborhoods of the active nodes. This in turn yields the  $\tilde{O}(m)$  total memory bound. Property (b) holds for algorithms whose running time is dominated by  $O(\log \Delta)$ . Given that the degree threshold in Property (a) falls as  $\Delta^{1/2^i}$ , the running time of each stage falls by a factor of 2.

► **Lemma 11.** *Suppose that Alg is a state-congested, degree-ordered algorithm. Consider a phase of  $\ell - 1$  rounds  $t + 1, t + 2, \dots, t + \ell - 1$  with a Stage  $i$ . If  $\ell$  satisfies*

$$\left( \sum_{\tau=t+1}^{t+\ell} \tilde{A}_\tau \log n \right)^\ell \leq \min \left\{ n^{\varepsilon/2}, \Delta^{1/2^\ell} \right\}, \quad (2)$$

*then this phase can be simulated in  $O(\log \ell)$  rounds in the low-memory MPC model with a total of  $\tilde{O}(m)$  memory over all the machines.*

Finally, if Alg is a state-congested algorithm that is  $\alpha$ -sparse and degree-ordered, we obtain the following Simulation Theorem that guarantees an  $\tilde{O}(m)$  total memory usage.

► **Theorem 12** (Sample-and-Gather Theorem v2). *Let Alg be a state-congested,  $\alpha$ -sparse, degree-ordered algorithm that completes in  $R$  rounds. Let  $\alpha' = \alpha \cdot \log^2 n$ . Then Alg can be simulated in the MPC model with  $O(n^\varepsilon)$  memory per machine, for constant  $0 < \varepsilon < 1$  and  $\tilde{O}(m)$  total memory, in  $O\left(R \log \log \Delta / \sqrt{\log_{\alpha'} \Delta}\right)$  rounds.*

### 3 Fast 2-Ruling Set Algorithms

Our 2-ruling set algorithms consist of 3 parts. In Part 1, we sparsify the input graph, in Part 2 we “shatter” the graph still active after Part 1, and in Part 3 we deterministically finish off the computation. Part 1 is a modification of SPARSIFY, a CONGEST model algorithm due to Kothapalli and Pemmaraju [26]; Part 2 is a sparsified MIS algorithm, also in the CONGEST model, due to Ghaffari [15, 16]. Our main contribution in this section is to show that these algorithms are state-congested,  $\alpha$ -sparse for small  $\alpha$ , and degree-ordered. As a result, we can apply the Sample-and-Gather Simulation Theorems (Theorems 9 and 12) to these algorithms to obtain fast low-memory MPC algorithms. Part 3 – in which we finish off the computation – is easy to directly implement in the MPC model.

#### 3.1 Simulating Sparsify in low-memory MPC

Algorithm 1 is a modified version of the SPARSIFY algorithm of Kothapalli and Pemmaraju [26]. The algorithm computes a “sparse” set of vertices  $U$  that dominates all the vertices in the graph (i.e.  $\text{Nbr}^+(U) = V$ , see Lemma 13). In each iteration, “high degree” nodes and their neighbors are sampled and the sampled nodes are added to  $U$ . In successive iterations, the threshold for being a high degree node falls by a factor  $f$  and the sampling probability grows by a factor  $f$ . The neighbors of the nodes that successfully join  $U$  are deactivated. The parameter  $f$  takes on different values in different instantiations of this algorithm, though always satisfying  $\log f = \log^\delta \Delta$  for some constant  $0 < \delta < 1$ . For example,  $f$  is set to  $2^{(\log \Delta)^{2/3}}$  (respectively,  $2^{(\log \Delta)^{1/2}}$ ) to obtain Theorem 19 part (i) (respectively, part (ii)).

DEORDEREDSPARSIFY fits nicely within the framework of the Sample-and-Gather Simulation Theorems from Section 2. The state of each vertex stays small throughout the algorithm (just ID plus  $O(1)$  bits), making DEORDEREDSPARSIFY state-congested. The activity level in any iteration is bounded by  $O(f \log n)$ , because we show in Lemma 13 that in any neighborhood only  $O(f \log n)$  vertices are sampled whp and only these sampled vertices need be active in that iteration. Furthermore, since the sampling probability grows by a factor  $f$  in each iteration, the estimated neighborhood activity levels also grow by a factor  $f$ , as we consider future iterations of DEORDEREDSPARSIFY. As shown in Lemma 13, this makes DEORDEREDSPARSIFY  $f$ -sparse. In the SPARSIFY algorithm [26] *all* nodes,

■ **Algorithm 1** `DEGORDEREDSPARSIFY( $G, f$ )`.

---

```

1  $U \leftarrow \emptyset$ 
2  $V_0 \leftarrow V$  // Initially all nodes are active
3 for  $i = 1$  to  $\lceil \log_f \Delta \rceil$  do
4   Let  $H_i$  be the nodes in  $V_{i-1}$  with degree at least  $\Delta/f^i$  in  $G[V_{i-1}]$ 
5   Each node in  $\text{Nbr}^+(H_i) \cap V_{i-1}$  joins  $U_i$  with probability  $f^i \cdot c \ln n / \Delta$ , where  $c$  is a
   fixed constant
6    $V_i \leftarrow V_{i-1} \setminus \text{Nbr}^+(U_i)$  // Nodes with at least one neighbor in  $U_i$ 
   deactivate themselves
7    $U \leftarrow U \cup U_i$ 
8 end
9 return  $U$ 

```

---

independent of their degrees, sample themselves (as in Line 5). Here, in order to make Algorithm `DEGORDEREDSPARSIFY` degree-ordered, we make a small modification and permit only high degree nodes and their neighbors to sample themselves. As we show in Lemma 13, the algorithm continues to behave as before, but is now degree-ordered.

► **Lemma 13.** *Given a graph  $G = (V, E)$  and a parameter  $f > 3$ , a subset  $U \subseteq V$  can be computed in  $O(\log_f \Delta)$  rounds such that for every  $v \in V$ ,  $N^+(v) \cap U \neq \emptyset$ , and for every  $v \in U$ ,  $\deg_U(v) \leq 2cf \ln n$ , with probability at least  $1 - n^{-c+2}$ .*

It is easy to see that Algorithm `DEGORDEREDSPARSIFY( $G, f$ )` can be implemented in the `CONGEST` model in  $O(\log_f \Delta)$  rounds because each iteration of the **for**-loop takes  $O(1)$  rounds in `CONGEST`. Furthermore, since each node can update its state by simply knowing if it or a neighbor has joined set  $U_i$ , the update function at each node is separable (see Definition 1). Therefore, `DEGORDEREDSPARSIFY( $G, f$ )` can be faithfully simulated in the low-memory MPC model in  $O(\log_f \Delta)$  rounds.

We now show that Algorithm `DEGORDEREDSPARSIFY` has the three properties needed for round compression via our Simulation Theorems and this leads to a substantial speedup.

► **Lemma 14.** *Algorithm `DEGORDEREDSPARSIFY( $G, f$ )` is a state-congested,  $f$ -sparse, degree-ordered algorithm.*

Using Theorem 9 and Theorem 12, we obtain the following theorem.

► **Theorem 15.** *Algorithm `DEGORDEREDSPARSIFY( $G, f$ )` can be implemented in the low-memory MPC model in (i)  $O\left(\frac{\log_f \Delta}{\sqrt{\log_f n}} \log \log n\right)$  rounds whp using  $\tilde{O}(m + n^{1+\varepsilon})$  total memory and (ii)  $O(\sqrt{\log_f \Delta} \cdot \log \log \Delta)$  rounds whp using  $\tilde{O}(m)$  total memory.*

### 3.2 Simulating Sparsified Graph Shattering in low-memory MPC

Distributed graph shattering has become an important algorithmic technique for symmetry breaking problems [5, 16, 20]. In this section, we use a sparsified graph shattering algorithm due to Ghaffari [16] to process the graph  $G[U]$  returned by `DEGORDEREDSPARSIFY`. The output of the shattering algorithm consists of an independent set  $I \subseteq U$  such that the graph induced by the remaining set of vertices  $S = U \setminus \text{Nbr}^+(I)$  contains only small connected components.

Ghaffari's sparsified shattering algorithm [16] is shown in Algorithm 2. At the start of each round  $t$ , each node  $v$  has a *desire-level*  $p_t(v)$  for joining the independent set  $I$ , and initially this is set to  $p_1(v) = 1/2$ . The independent set  $I$  is also initialized to the empty set. The algorithm runs in phases, with each phase having  $\ell := \sqrt{\delta \log n}/10$  rounds for a small constant  $\delta$ .

Several aspects of the algorithm make it nicely fit the Sample-and-Gather framework from Section 2. We now point these out. (i) The desire-level  $p_\tau(u)$  for  $t + 1 \leq \tau \leq t + \ell$  can be viewed the probability of sampling  $u$ ; after the initial communication amongst neighbors (Line 1), only sampled nodes send messages (beeps) and all other nodes remain silent. (ii) The quantity  $d_{t+1}(u)$  is identical to the activity level  $A_{t+1}(u)$  in  $u$ 's neighborhood, defined in Section 2. (iii) Nodes with a high activity level, i.e.,  $d_{t+1}(u) \geq 2\sqrt{\log n/5}$  (aka super-heavy nodes), are oblivious nodes and are therefore excluded in the definition of  $A_{t+1}$ . As a result  $A_{t+1} \leq 2\sqrt{\log n/5}$ . (iv) In each iteration in a phase, the sampling probability grows by a factor of at most 2 (Line 8). This implies that the estimated activity levels grow by a factor of 2 in future rounds.

■ **Algorithm 2** SHATTER( $G$ ): (one phase, starting at iteration  $t + 1$ ).

---

```

1 Each node  $u$  sends its current desire-level  $p_{t+1}(u)$  to all its neighbors
2 Each node  $u$  computes  $d_{t+1}(u) = \sum_{v \in \text{Nbr}(u)} p_{t+1}(v)$ 
3 If node  $u$  has  $d_{t+1}(u) \geq 2\sqrt{\log n/5}$  then  $u$  is called a super-heavy node
4  $\ell = \sqrt{\delta \log n}/10$  ; //  $\delta$  is a small constant
5 for  $\tau = t + 1, t + 2, \dots, t + \ell$  iterations do
    // Round 1
6 Each node  $u$  beeps with probability  $p_\tau(u)$  and remains silent otherwise.
7 Node  $u$  is added to  $I$  if it is not super-heavy, it beeps, and none of its neighbors
  beep
8 Node  $u$  sets  $p_{\tau+1}(u)$  as follows:
    
$$p_{\tau+1}(u) = \begin{cases} p_\tau(u)/2 & \text{if } u \text{ is super-heavy, or a neighbor of } u \text{ beeps} \\ \min\{1/2, 2 \cdot p_\tau(u)\} & \text{otherwise} \end{cases}$$

    // Round 2
9 Node  $u$  beeps if it joins  $I$  in this iteration.
10 Neighbors of node  $u$  that are not in  $I$  become inactive on hearing the beep from  $u$ 
11 end

```

---

The first four steps of Algorithm 2 do not fit into the Sample-and-Gather framework since each node needs to send its  $p_{t+1}$  value to its neighbors. But the nodes are computing  $d_{t+1}(u) = \sum_{v \in \text{Nbr}(u)} p_{t+1}(v)$  which is a separable function (sum). Therefore, we can implement the first two steps in  $O(1/\varepsilon)$  rounds using Lemma 2, and use the Sample-and-Gather framework to simulate the **for**-loop of the algorithm. These observations are formalized in the lemma below to show that SHATTER is 2-sparse. Additionally, the lemma shows that the algorithm is state-congested.

► **Lemma 16.** *Algorithm 2 is a state-congested algorithm whose **for**-loop is 2-sparse.*

A total of  $O(\log \Delta / \sqrt{\log n})$  repeated applications of SHATTER (i.e. a total of  $O(\log \Delta)$  iterations) suffice to shatter the graph into small-sized components [16, 21].

Using Lemma 16 and Theorem 9, we obtain the following lemma that shows that SHATTER can be simulated efficiently in the low-memory MPC model.

► **Lemma 17.** *We can simulate a total  $O(\log \Delta)$  iterations of Algorithm SHATTER in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory in  $O\left(\frac{\log \Delta \cdot \log \log n}{\sqrt{\log n}}\right)$  rounds whp.*

Ghaffari and Uitto [21] present a variant of Algorithm SHATTER and show (in Theorem 3.7) that this variant can be simulated in  $O(\sqrt{\log \Delta} \cdot \log \log \Delta)$  rounds in the low-memory MPC model, while using only  $\tilde{O}(m)$  total memory. While they describe their MPC implementation from scratch, this MPC implementation can also be obtained by applying our Sample-and-Gather Theorem (specifically, Theorem 12). It can be shown that this variant is state-congested and has the same sparsity property as Algorithm SHATTER, i.e., it is 2-sparse. Furthermore, it can also be made degree-ordered by simply processing nodes in degree buckets  $(\Delta^{1/2^i}, \Delta^{1/2^{i-1}}]$ , in the order  $i = 1, 2, \dots, O(\log \log \Delta)$ .

► **Lemma 18** (Ghaffari-Uitto [21]). *There is a variant of Algorithm SHATTER can be simulated in the low-memory MPC model with  $\tilde{O}(m)$  total memory in  $O(\sqrt{\log \Delta} \cdot \log \log \Delta)$  rounds whp.*

### 3.3 Finishing off the 2-ruling set computation

After applying DEORDEREDSPARSIFY to the input graph  $G = (V, E)$  and then SHATTER to the subgraph  $G[U]$ , induced by the subset  $U \subseteq V$  output by DEORDEREDSPARSIFY, we are left with a number of small-sized components. Ghaffari and Uitto [21, Theorem 3.7] show that given the properties that the remaining graph has after SHATTER, it is possible to simply (and deterministically) gather each component at a machine and find an MIS of the component locally in  $O(\sqrt{\log \log n})$  rounds in the low-memory MPC model using  $\tilde{O}(m)$  memory. Applying this “finishing off” computation completes our 2-ruling set algorithm. The output of the algorithm is the union of the independent set output by SHATTER and the independent set output by the “finishing off” computation.

► **Theorem 19.** *A 2-ruling set can be computed whp in the low-memory MPC model in*

- (i)  $O((\log \Delta)^{1/6} \log \log n)$  rounds using  $\tilde{O}(m + n^{1+\varepsilon})$  total memory and in
- (ii)  $O((\log \Delta)^{1/4} \log \log \Delta + \sqrt{\log \log n} \log \log \Delta)$  rounds using  $\tilde{O}(m)$  total memory.

► **Remark.** We note that by just running SHATTER on an input graph followed by the “finishing off” computation, we get an MIS of the input graph. So our approach yields MIS algorithms in the low-memory MPC model via the Sample-and-Gather Simulation Theorems.

► **Theorem 20.** *An MIS of a graph  $G$  can be found in the low-memory MPC model in:*

- (i)  $O\left(\frac{\log \Delta \cdot \log \log n}{\sqrt{\log n}} + \sqrt{\log \log n}\right)$  rounds whp using  $\tilde{O}(m + n^{1+\varepsilon})$  total memory and
- (ii)  $O(\sqrt{\log \Delta} \log \log \Delta + \sqrt{\log \log n})$  rounds whp using  $\tilde{O}(m)$  total memory.

As far as we know, the MIS result for the  $\tilde{O}(m + n^{1+\varepsilon})$  total memory setting is new, but the result for the  $\tilde{O}(m)$  total memory setting simply recovers the result from [21].

## 4 Fast $\beta$ -ruling Set Algorithms

We now extend the 2-ruling set low-memory MPC algorithm in the previous section to obtain a  $\beta$ -ruling set low-memory MPC algorithm for any integer  $\beta \geq 2$ . The overall idea is to repeatedly use Algorithm DEORDEREDSPARSIFY, as in [8]. We start by running a

low-memory MPC implementation of DEORDEREDSPARSIFY with a parameter  $f_1$ ; this call returns a set of nodes  $S_1$ . Once this phase ends, the remaining graph  $G[S_1]$  has degree at most  $O(f_1 \cdot \log n)$ , by Lemma 13. We then run DEORDEREDSPARSIFY on the graph  $G[S_1]$  with a parameter  $f_2$  and this yields a set of nodes  $S_2$ . This process continues for  $\beta - 1$  phases at the end of which the graph  $G[S_{\beta-1}]$  has a maximum degree  $O(f_{\beta-1} \cdot \log n)$ . We now proceed to run a low-memory MPC implementation of an MIS algorithm on  $G[S_{\beta-1}]$ .

The correctness of the  $\beta$ -ruling set algorithm can be noted from Lemma 13. The set  $S_i$  covers all the nodes in  $S_{i-1}$  which means that all the nodes in  $S_0 = V$  are at most  $\beta - 1$  hops away from the nodes in  $S_{\beta-1}$ . Therefore all the nodes of  $V$  are at most  $\beta$  hops away from the MIS  $C$  of  $G[S_{\beta-1}]$ . This means that the set  $C$  that the above technique returns is a  $\beta$ -ruling set of  $G$ . In the following, we analyze the round complexity of the  $\beta$ -ruling set algorithm in the low-memory MPC model.

► **Lemma 21.** *Let  $f_0 = \Delta$ . The  $\beta$ -ruling set algorithm runs in*

$$O\left(\left(\sum_{i=1}^{\beta-1} \frac{\log(f_{i-1} \log n)}{\sqrt{\log f_i \cdot \log n}} + \frac{\log(f_{\beta-1} \log n)}{\sqrt{\log n}}\right) \log \log n\right) \quad (3)$$

*rounds whp in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory.*

► **Lemma 22.** *Let  $f_0 = \Delta$ . The  $\beta$ -ruling set algorithm runs in*

$$O\left(\left(\sum_{i=1}^{\beta-1} \sqrt{\frac{\log(f_{i-1} \log n)}{\log f_i}} + \sqrt{\log(f_{\beta-1} \log n)}\right) \log \log \Delta + \sqrt{\log \log n}\right) \quad (4)$$

*rounds whp in the low-memory MPC model with  $\tilde{O}(m)$  total memory.*

We now instantiate the parameters  $f_1, f_2, \dots, f_{\beta-1}$  so as to minimize the running times in Lemmas 21 and 22. This leads to the following corollaries.

► **Theorem 23.** *A  $\beta$ -ruling set of a graph  $G$  can be found whp in the low-memory MPC model in*

- (i)  $O\left(\beta \cdot \log^{1/(2^{\beta+1}-2)} \Delta \cdot \log \log n\right)$  rounds with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory and in
- (ii)  $O\left(\beta \cdot \left(\log^{1/2\beta} \Delta \cdot \log \log \Delta + \sqrt{\log \log n}\right) \cdot \log \log \Delta\right)$  rounds with  $\tilde{O}(m)$  total memory.

#### 4.1 $\beta$ -ruling sets in $O(\text{polyloglog}(n))$ rounds

As mentioned in the Introduction, this research is partly motivated by the question of whether ruling set problems can be solved in the low-memory MPC model in  $O(\text{polyloglog}(n))$  rounds. Using our results we identify two interesting circumstances under which  $\beta$ -ruling sets can be computed in the low-memory MPC model in  $O(\text{polyloglog}(n))$  rounds. First, because the running time in Theorem 23 part (i) has an inverse exponential dependency on  $\beta$ , we get the following corollary.

► **Corollary 24.** *For  $\beta \in \Omega(\log \log \log \Delta)$ , a  $\beta$ -ruling set of a graph  $G$  can be computed in  $O(\beta \log \log n)$  rounds whp in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory.*

Second, we can also show that for graphs with bounded  $\Delta$ , we can compute a  $\beta$ -ruling set in  $O(\beta \log \log n)$  rounds, however this bound increases quickly with  $\beta$ , giving us the following corollary.

► **Corollary 25.** *If we have that  $\Delta = O\left(2^{\log^{1-\frac{1}{2\beta}} n}\right)$ , then a  $\beta$ -ruling set can be computed in  $O(\beta \log \log n)$  rounds whp in the low-memory MPC model with  $\tilde{O}(m + n^{1+\varepsilon})$  total memory.*

---

**References**

---

- 1 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, December 1986. doi:10.1016/0196-6774(86)90019-2.
- 2 Sepehr Assadi. Simple round compression for parallel vertex cover. *CoRR*, abs/1709.04599, 2017. arXiv:1709.04599.
- 3 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for  $(\Delta + 1)$  vertex coloring. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.
- 4 Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively parallel algorithms for finding well-connected components in sparse graphs. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 461–470, 2019. doi:10.1145/3293611.3331596.
- 5 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016. doi:10.1145/2903137.
- 6 Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 481–490, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331609.
- 7 Soheil Behnezhad, Mohammadtaghi Hajiaghayi, and David G. Harris. Exponentially Faster Massively Parallel Maximal Matching. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, volume 2019-November, pages 1637–1649, 2019. doi:10.1109/FOCS.2019.00096.
- 8 Tushar Bisht, Kishore Kothapalli, and Sriram V. Pemmaraju. Brief announcement: Super-fast t-ruling sets. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 379–381. ACM, 2014. doi:10.1145/2611462.2611512.
- 9 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of  $(\delta + 1)$  coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 471–480, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331607.
- 10 Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional lower bounds for adaptive massively parallel computation. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, page 141–151, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3350755.3400230.
- 11 Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, August 2015. doi:10.14778/2824032.2824077.
- 12 Artur Czumaj, Peter Davies, and Merav Parter. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space, 2019. arXiv:1912.05390.
- 13 Artur Czumaj, Slobodan Mitrovic, Jakub Łącki, Krzysztof Onak, Aleksander Mądry, and Piotr Sankowski. Round compression for parallel matching algorithms. *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 471–484, 2018. doi:10.1145/3188745.3188764.
- 14 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. doi:10.1145/1327452.1327492.
- 15 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 270–277, USA, 2016. Society for Industrial and Applied Mathematics.

- 16 Mohsen Ghaffari. Distributed MIS via all-to-all communication. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, Part F129314:141–150, 2017. doi:10.1145/3087801.3087830.
- 17 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 129–138, 2018. doi:10.1145/3212734.3212743.
- 18 Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. *CoRR*, 2020. arXiv:2002.09610.
- 19 Mohsen Ghaffari, Ce Jin, and Daan Nilis. A massively parallel algorithm for minimum weight vertex cover. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 259–268. ACM, 2020. doi:10.1145/3350755.3400260.
- 20 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, volume 2019-November, pages 1650–1663, 2019. doi:10.1109/FOCS.2019.00097.
- 21 Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653, July 2019. doi:10.1137/1.9781611975482.99.
- 22 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '18, page 43–52, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3210377.3210386.
- 23 James W. Hegeman, Sriram V. Pemmaraju, and Vivek Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. *CoRR*, abs/1408.2071, 2014. arXiv:1408.2071.
- 24 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948, Philadelphia, PA, January 2010. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973075.76.
- 25 Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju. Sample-and-gather: Fast ruling set algorithms in the low-memory MPC model, 2020. arXiv:2009.12477.
- 26 Kishore Kothapalli and Sriram V. Pemmaraju. Super-fast 3-ruling sets. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPICs*, pages 136–147. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.FSTTCS.2012.136.
- 27 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, page 300–309, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1011767.1011811.
- 28 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2), March 2016. doi:10.1145/2742012.
- 29 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, February 1992. doi:10.1137/0221015.
- 30 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in  $o(\log \log n)$  communication rounds. In *SPAA*, pages 94–100, 2003. doi:10.1145/777412.777428.
- 31 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi:10.1137/0215074.
- 32 Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In

- Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 135–146, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1807167.1807184.
- 33 Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPICs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.DISC.2018.40.
  - 34 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.
  - 35 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *J. ACM*, 65(6), November 2018. doi:10.1145/3232536.
  - 36 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363, 2020. doi:10.1145/3357713.3384298.
  - 37 Grigory Yaroslavtsev and Adithya Vadapalli. Massively parallel algorithms and hardness for single-linkage clustering under  $\ell_p$  distances. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5596–5605. PMLR, 2018. URL: <http://proceedings.mlr.press/v80/yaroslavtsev18a.html>.
  - 38 Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. doi:10.1145/2934664.

# On Parity Decision Trees for Fourier-Sparse Boolean Functions

Nikhil S. Mande

Georgetown University, Washington, DC, USA

<http://nikhil.georgetown.domains/>

[nikhil.mande@georgetown.edu](mailto:nikhil.mande@georgetown.edu)

Swagato Sanyal

Indian Institute of Technology Kharagpur, India

[swagato@cse.iitkgp.ac.in](mailto:swagato@cse.iitkgp.ac.in)

---

## Abstract

---

We study parity decision trees for Boolean functions. The motivation of our study is the log-rank conjecture for XOR functions and its connection to Fourier analysis and parity decision tree complexity. Our contributions are as follows. Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be a Boolean function with Fourier support  $\mathcal{S}$  and Fourier sparsity  $k$ .

- We prove via the probabilistic method that there exists a parity decision tree of depth  $O(\sqrt{k})$  that computes  $f$ . This matches the best known upper bound on the parity decision tree complexity of Boolean functions (Tsang, Wong, Xie, and Zhang, FOCS 2013). Moreover, while previous constructions (Tsang et al., FOCS 2013, Shpilka, Tal, and Volk, Comput. Complex. 2017) build the trees by carefully choosing the parities to be queried in each step, our proof shows that a naive sampling of the parities suffices.
- We generalize the above result by showing that if the Fourier spectra of Boolean functions satisfy a natural “folding property”, then the above proof can be adapted to establish existence of a tree of complexity polynomially smaller than  $O(\sqrt{k})$ . More concretely, the folding property we consider is that for most distinct  $\gamma, \delta$  in  $\mathcal{S}$ , there are at least a polynomial (in  $k$ ) number of pairs  $(\alpha, \beta)$  of parities in  $\mathcal{S}$  such that  $\alpha + \beta = \gamma + \delta$ . We make a conjecture in this regard which, if true, implies that the communication complexity of an XOR function is bounded above by the fourth root of the rank of its communication matrix, improving upon the previously known upper bound of square root of rank (Tsang et al., FOCS 2013, Lovett, J. ACM. 2016).
- Motivated by the above, we present some structural results about the Fourier spectra of Boolean functions. It can be shown by elementary techniques that for any Boolean function  $f$  and all  $(\alpha, \beta)$  in  $\binom{\mathcal{S}}{2}$ , there exists another pair  $(\gamma, \delta)$  in  $\binom{\mathcal{S}}{2}$  such that  $\alpha + \beta = \gamma + \delta$ . One can view this as a “trivial” folding property that all Boolean functions satisfy. Prior to our work, it was conceivable that for all  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$ , there exists exactly one other pair  $(\gamma, \delta) \in \binom{\mathcal{S}}{2}$  with  $\alpha + \beta = \gamma + \delta$ . We show, among other results, that there must exist several  $\gamma \in \mathbb{F}_2^n$  such that there are at least three pairs of parities  $(\alpha_1, \alpha_2) \in \binom{\mathcal{S}}{2}$  with  $\alpha_1 + \alpha_2 = \gamma$ . This, in particular, rules out the possibility stated earlier.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Oracles and decision trees

**Keywords and phrases** Parity decision trees, log-rank conjecture, analysis of Boolean functions, communication complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.29

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2008.00266>.

**Funding** *Swagato Sanyal*: Supported by an ISIRD Grant from Sponsored Research and Industrial Consultancy, IIT Kharagpur.

**Acknowledgements** We thank Prahladh Harsha, Srikanth Srinivasan, Sourav Chakraborty and Manaswi Paraashar for useful discussions.



© Nikhil S. Mande and Swagato Sanyal;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 29; pp. 29:1–29:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

The log-rank conjecture [6] is a fundamental unsolved question in communication complexity that states that the deterministic communication complexity of a Boolean function is polynomially related to the logarithm of the rank (over real numbers) of its communication matrix. The importance of the conjecture stems from the fact that it proposes to characterize communication complexity, which is an interactive complexity measure, by the rank of a matrix which is a traditional and well-understood algebraic measure. In this work we focus on the important and well-studied class of XOR functions. Consider a two-party function  $F : \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \{-1, 1\}$  whose value on any input  $(x, y)$  depends only on the bitwise XOR of  $x$  and  $y$ , i.e., there exists a function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  such that for each  $(x, y) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ ,  $F(x, y) = f(x \oplus y)$ . Such a function  $F$  is called an XOR function, and is denoted as  $F = f \circ \oplus$ . The log-rank conjecture and communication complexity of such an XOR function  $F$  has interesting connections with the Fourier spectrum of  $f$ . For example, it is known that the rank of the communication matrix of  $F$  equals the Fourier sparsity of  $f$  (henceforth referred to as  $k$ ) [2]. The natural randomized analogue of the log-rank conjecture is the log-approximate-rank conjecture [5], which was recently refuted by Chattopadhyay, Mande, and Sherif [3]. The quantum analogue of the log-rank conjecture was subsequently also refuted by Sinha and de Wolf [12] and Anshu, Boddu, and Touchette [1]. It is worth noting that an XOR function was used to refute these conjectures.

To design a cheap communication protocol for  $F$ , an approach adopted by many works [11, 14, 9] is to design a small-depth parity decision tree (henceforth referred to as PDT) for  $f$ , and having a communication protocol simulate the tree; it is easy to see that the parity of a subset of bits of the string  $x \oplus y$  can be computed by the communicating parties by interchanging two bits. The parity decision tree complexity (henceforth referred to as  $\text{PDT}(\cdot)$ ) of  $f$  thus places an asymptotic upper bound on the communication complexity of  $F$ . The work of Hatami, Hosseini and Lovett [4] shows that this approach is polynomially tight; they showed that  $\text{PDT}(f)$  is polynomially related to the deterministic communication complexity of  $F$ . In light of this, the log-rank conjecture for XOR functions  $F = f \circ \oplus$  is readily seen to be equivalent to  $\text{PDT}(f)$  being polylogarithmic in  $k$ .

However, we are currently very far from achieving this goal. Lovett [7] showed that the deterministic communication complexity of any Boolean function  $F$  is bounded above by  $O(\sqrt{\text{rank}(F)} \log \text{rank}(F))$ . In particular, this implies that the deterministic communication complexity of  $F = f \circ \oplus$  is  $O(\sqrt{k} \log k)$ . Tsang et al. [14] showed that  $\text{PDT}(f) = O(\sqrt{k})$  (a quantitatively weaker bound was shown in a simultaneous and independent work of Shpilka et al. [11]). In addition to bounding  $\text{PDT}(f)$  instead of the communication complexity of  $F$ , Tsang et al. achieved a quantitative improvement by a logarithmic factor over Lovett's bound for the class of XOR functions. Sanyal [10] showed that the simultaneous communication complexity of  $F$  (characterized by the Fourier dimension of  $f$ ) is bounded above by  $O(\sqrt{k} \log k)$ , and is tight (up to the  $\log k$  factor) for the addressing function.

In this work we derive new understanding about the structure of Fourier spectra of Boolean functions. Aided by this insight we reprove the  $O(\sqrt{k})$  upper bound on  $\text{PDT}(f)$  (see Sections 3.1 and 3.2). We conditionally improve this bound by a polynomial factor, assuming a “folding property” of the Fourier spectra of Boolean functions (see Section 3.3). To prove these results, we make use of a simple necessary condition for a function to be Boolean (see Proposition 5). While we show that it is not a sufficient condition (see Theorem 27), it does enable us to prove the above results. In these proofs, we use Proposition 5 in conjunction with probabilistic and combinatorial arguments. Finally, we make progress

towards establishing the folding property (see Section 3.4). To prove these results, we use the well-known characterization of Boolean functions given by two conditions, namely Parseval's identity (Equation (2)) and a condition attributed to Titsworth (Equation (3)), in conjunction with combinatorial arguments.

## 1.1 Organization of this paper

In Section 2 we review some preliminaries and introduce the notation that we use in this paper. In this section we also introduce definitions and concepts that are needed to state our results formally. In Section 3 we motivate and formally state our results, and discuss proof techniques. The formal proofs of our main results can be found in Sections 4 and 5 of this paper, and in Sections 5 and 6 of the full version of this paper [8].

## 2 Notation and preliminaries

All logarithms in this paper are taken with base 2. We use the phrase “ $k$  is sufficiently large” to mean that there exists a universal constant  $C > 0$  such that  $k > C$ . As is standard, we use the notation  $f(n) = \tilde{O}(h(n))$  ( $f(n) = \tilde{\Theta}(\cdot)$ ,  $f(n) = \tilde{\Omega}(\cdot)$ ) to convey that there exists a constant  $c \geq 0$  such that that  $f(n) = O(h(n) \log^c h(n))$  ( $f(n) = \Theta(h(n) \log^c h(n))$ ,  $f(n) = \Omega(h(n) \log^c h(n))$ , respectively). We use the notation  $[n]$  to denote the set  $\{1, 2, \dots, n\}$ . For any set  $S$ , we use the notation  $\binom{S}{2}$  to denote the set of all subsets of  $S$  of size exactly 2. We abuse notation and denote a generic element of  $\binom{S}{2}$  as  $(a, b)$  rather than  $\{a, b\}$ . When we use the notation  $\mathbb{E}_{x \in X}[\cdot]$ , the underlying distribution corresponds to  $x$  being sampled uniformly at random from  $X$ . For  $a \in \mathbb{F}_2$ , we let  $a^n$  denote the  $n$ -bit string  $(a, a, \dots, a)$ . We use the symbol “+” to denote both coordinate-wise addition over  $\mathbb{F}_2$  as well as addition over reals; the meaning in use will be clear from context. For sets  $A, B \subseteq \mathbb{F}_2^n$ ,  $A + B$  denotes the *sumset* defined by  $\{\alpha + \beta \mid \alpha \in A, \beta \in B\}$ . For a set  $A \subseteq \mathbb{F}_2^n$  and  $\gamma \in \mathbb{F}_2^n$ , we denote by  $A + \gamma$  the set  $A + \{\gamma\}$ . The above convention also extends to the symbol “ $\sum$ ”. For a set of vectors  $\Gamma \in \mathbb{F}_2^n$ , we define  $\text{span } \Gamma$  to be the set of all  $\mathbb{F}_2$ -linear combinations of vectors in  $\Gamma$ , i.e.,  $\text{span } \Gamma = \left\{ \sum_{\gamma \in \Gamma} c_\gamma \cdot \gamma \mid c_\gamma \in \mathbb{F}_2 \text{ for } \gamma \in \Gamma \right\}$ .

Consider the vector space of functions from  $\mathbb{F}_2^n$  to  $\mathbb{R}$ , equipped with the following inner product.

$$\langle f, g \rangle := \mathbb{E}_{x \in \mathbb{F}_2^n} [f(x)g(x)] = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} f(x)g(x).$$

Let  $x = (x_1, \dots, x_n) \in \mathbb{F}_2^n$ . For each  $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{F}_2^n$ , define  $\alpha(x) := \sum_{i=1}^n \alpha_i x_i \pmod{2}$ , and the associated *character*  $\chi_\alpha : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  by  $\chi_\alpha(x) := (-1)^{\alpha(x)}$ . Observe that  $\chi_\alpha(x)$  is the  $\pm 1$ -valued parity of the bits  $\{x_i \mid \alpha_i = 1\}$ ; due to this we will also refer to characters as parities. The set of parities  $\{\chi_\alpha \mid \alpha \in \mathbb{F}_2^n\}$  forms an orthonormal (with respect to the above inner product) basis for this vector space. Hence, every function  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$  can be uniquely written as  $f = \sum_{\alpha \in \mathbb{F}_2^n} \hat{f}(\alpha) \chi_\alpha$ , where  $\hat{f}(\alpha) = \langle f, \chi_\alpha \rangle = \mathbb{E}_{x \in \mathbb{F}_2^n} [f(x) \chi_\alpha(x)]$ . The coefficients  $\{\hat{f}(\alpha) \mid \alpha \in \mathbb{F}_2^n\}$  are called the Fourier coefficients of  $f$ .

For any function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  and any set  $A \subseteq \mathbb{F}_2^n$ , define the function  $f|_A : A \rightarrow \{-1, 1\}$  by  $f|_A(x) = f(x)$  for all  $x \in A$ . In other words,  $f|_A$  denotes the restriction of  $f$  to  $A$ .

Throughout this paper, for any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ , we denote by  $\mathcal{S}$  the Fourier support of  $f$ , i.e.  $\mathcal{S} = \left\{ \alpha \in \mathbb{F}_2^n \mid \hat{f}(\alpha) \neq 0 \right\}$ . We also denote by  $k$  the Fourier sparsity of  $f$ , i.e.  $k = |\mathcal{S}|$ . The dependence of  $\mathcal{S}$  and  $k$  on  $f$  is suppressed and the underlying function will be clear from context.

## 29:4 On Parity Decision Trees for Fourier-Sparse Boolean Functions

The representation of Fourier coefficients as an expectation (over  $x \in \mathbb{F}_2^n$ ) immediately yields the following observation about granularity of Fourier coefficients of Boolean functions.

► **Observation 1.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. Then, for all  $\alpha \in \mathbb{F}_2^n$ ,  $\widehat{f}(\alpha)$  is an integral multiple of  $1/2^n$ .*

We next define plateaued functions.

► **Definition 2 (Plateaued functions).** *A Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  is said to be plateaued if there exists  $x \in \mathbb{R}$  such that  $\widehat{f}(\alpha) \in \{0, x, -x\}$  for all  $\alpha \in \mathbb{F}_2^n$ .*

Next we define the addressing function.

► **Definition 3 (Addressing function).** *Let  $k$  be an even power of 2. The addressing function  $\text{ADD}_k : \mathbb{F}_2^{\frac{1}{2} \log k + \sqrt{k}} \rightarrow \{-1, 1\}$  is defined as*

$$\text{ADD}_k(x, y_1, \dots, y_{\sqrt{k}}) := (-1)^{y_{\text{int}(x)}},$$

where  $x \in \mathbb{F}_2^{\frac{1}{2} \log k}$ ,  $y_i \in \mathbb{F}_2$  for  $i = 1, \dots, \sqrt{k}$ , and  $\text{int}(x)$  is the unique integer in  $\{1, \dots, \sqrt{k}\}$  whose binary representation is  $x$ .

The Fourier sparsity of  $\text{ADD}_k$  can be verified to be  $k$ . We now define a notion of equivalence on elements of  $\binom{\mathcal{S}}{2}$ .

► **Definition 4.** *For any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ , we say a pair  $(\alpha_1, \alpha_2) \in \binom{\mathcal{S}}{2}$  is equivalent to  $(\alpha_3, \alpha_4) \in \binom{\mathcal{S}}{2}$  if  $\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4$ .*

In the above definition, if  $\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4 = \gamma$ , then we say that the pairs  $(\alpha_1, \alpha_2)$  and  $(\alpha_3, \alpha_4)$  fold in the direction  $\gamma$ . We also say that the elements  $\alpha_1, \alpha_2, \alpha_3$ , and  $\alpha_4$  participate in the folding direction  $\gamma$ . It is not hard to verify that the notion of equivalence defined above does indeed form an equivalence relation. We will denote by  $D_\gamma$  the equivalence class of pairs that fold in the direction  $\gamma$ , i.e.,

$$D_\gamma := \left\{ (\alpha, \beta) \in \binom{\mathcal{S}}{2} \mid \alpha + \beta = \gamma \right\}.$$

We suppress the dependence of  $D_\gamma$  on the underlying function  $f$ , which will be clear from context. Unless mentioned otherwise, these are the equivalence classes under consideration throughout this paper.

For any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ , we have for each  $x \in \mathbb{F}_2^n$ :

$$1 = f^2(x) = \sum_{\gamma \in \mathbb{F}_2^n} \left( \sum_{(\alpha_1, \alpha_2) \in \binom{\mathbb{F}_2^n}{2} : \alpha_1 + \alpha_2 = \gamma} \widehat{f}(\alpha_1) \widehat{f}(\alpha_2) \right) \chi_\gamma(x). \quad (1)$$

Matching the constant term of each side of the above identity we have

$$\sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)^2 = 1, \quad (2)$$

which is commonly referred to as Parseval's identity for Boolean functions. By matching the coefficient of each non-constant  $\chi_\gamma$  on each side of Equation (1) we obtain

$$\forall \gamma \neq 0^n, \quad \sum_{(\alpha_1, \alpha_2) \in \binom{\mathbb{F}_2^n}{2} : \alpha_1 + \alpha_2 = \gamma} \widehat{f}(\alpha_1) \widehat{f}(\alpha_2) = 0. \quad (3)$$

Equation (3) is attributed to Titsworth [13]. The following proposition is an easy consequence of Equation (3). It provides a necessary condition for a subset of  $\mathbb{F}_2^n$  to be the Fourier support of a Boolean function.

► **Proposition 5.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be a Boolean function. Then, for all  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$ , there exists  $(\gamma, \delta) \neq (\alpha, \beta) \in \binom{\mathcal{S}}{2}$  such that  $\alpha + \beta = \gamma + \delta$ . In other words,  $|D_{\alpha+\beta}| \geq 2$ .*

The Fourier  $\ell_1$ -norm of  $f$  is defined as  $\|\widehat{f}\|_1 := \sum_{\alpha \in \mathbb{F}_2^n} |\widehat{f}(\alpha)|$ . By the Cauchy-Schwarz inequality and Equation (2), we have

$$\|\widehat{f}\|_1 \leq \sqrt{k} \sqrt{\sum_{\alpha \in \mathbb{F}_2^n} \widehat{f}(\alpha)^2} = \sqrt{k}. \quad (4)$$

We next formally define parity decision trees.

A *parity decision tree* (PDT) is a binary tree whose leaf nodes are labeled in  $\{-1, 1\}$ , each internal node is labeled by a parity  $\chi_\alpha$  and has two outgoing edges, labeled  $-1$  and  $1$ . On an input  $x \in \mathbb{F}_2^n$ , the tree's computation proceeds from the root down as follows: compute  $\chi_\alpha(x)$  as indicated by the node's label and following the edge indicated by the value output, and continue in a similar fashion until reaching a leaf, at which point the value of the leaf is output. When the computation reaches a particular internal node, the PDT is said to *query* the parity label of that node. The PDT is said to compute a function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  if its output equals the value of  $f$  for all  $x \in \mathbb{F}_2^n$ . The parity decision tree complexity of  $f$ , denoted  $\text{PDT}(f)$  is defined as

$$\text{PDT}(f) := \min_{T: T \text{ is a PDT computing } f} \text{depth}(T).$$

## 2.1 Restriction to an affine subspace

In this section we discuss the effect of restricting a function  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$  to an affine subspace, on the Fourier spectrum of  $f$ .

► **Definition 6** (Affine subspace). *A set  $V \subseteq \mathbb{F}_2^n$  is called an affine subspace if there exist linearly independent vectors  $\ell_1, \dots, \ell_t \in \mathbb{F}_2^n$  and elements  $a_1, \dots, a_t \in \mathbb{F}_2$  such that  $V = \{x \in \mathbb{F}_2^n \mid \ell_i(x) = a_i \forall i \in \{1, \dots, t\}\}$ .  $t$  is called the co-dimension of  $V$ .*

Consider a set  $\Gamma := \{\gamma_1, \dots, \gamma_t\}$  of vectors in  $\mathbb{F}_2^n$ . Define  $\mathcal{G} := \text{span } \Gamma$ , and  $\mathcal{C} := \{\mathcal{G} + \beta \mid \beta \in \mathbb{F}_2^n, (\mathcal{G} + \beta) \cap \mathcal{S} \neq \emptyset\}$  to be the cosets of  $\mathcal{G}$  that have non-trivial intersection with  $\mathcal{S}$ . For each  $C \in \mathcal{C}$ , let  $\alpha(C)$  denote an arbitrary but fixed element in  $C \cap \mathcal{S}$ . In light of this, we write the Fourier transform of  $f$  as

$$f(x) = \sum_{C \in \mathcal{C}} \left( \sum_{\gamma \in \mathcal{G}} \widehat{f}(\alpha(C) + \gamma) \chi_\gamma(x) \right) \chi_{\alpha(C)}(x), \quad (5)$$

For any such fixed  $C$ , the value of the sum  $\sum_{\gamma \in \mathcal{G}} \widehat{f}(\alpha(C) + \gamma) \chi_\gamma(x)$  that appears in Equation (5) is determined by the values  $\gamma_1(x), \dots, \gamma_t(x)$ . Denote this sum by  $P_C(\gamma_1(x), \dots, \gamma_t(x))$ . For  $\mathbf{b} := (b_1, \dots, b_t) \in \mathbb{F}_2^t$ , let  $H_{\mathbf{b}}$  be the affine subspace  $\{x \in \mathbb{F}_2^n \mid \gamma_1(x) = b_1, \dots, \gamma_t(x) = b_t\}$ . It follows immediately that the Fourier transform of  $f|_{H_{\mathbf{b}}}$  is given by

$$f|_{H_{\mathbf{b}}}(x) = \sum_{C \in \mathcal{C}} P_C(b_1, \dots, b_t) \chi_{\alpha(C)}(x). \quad (6)$$

In particular, for each  $\mathbf{b}$ , the Fourier sparsity of  $f|_{H_{\mathbf{b}}}$  is bounded above by  $|\mathcal{C}|$ .

We note here that each element in  $\mathcal{S}$  is mapped to a unique element in  $\mathcal{C}$ . The elements of  $\mathcal{C}$  can thus be thought of as buckets that form a partition of  $\mathcal{S}$ . Keeping this view in mind we define the following.

## 29:6 On Parity Decision Trees for Fourier-Sparse Boolean Functions

► **Definition 7** (Bucket complexity). Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. Consider a set of vectors  $\Gamma = \{\gamma_1, \dots, \gamma_t\}$  in  $\mathbb{F}_2^n$ . Let  $\mathcal{G} := \text{span } \Gamma$ , and let  $\mathcal{C}$  denote the set of cosets of  $\mathcal{G}$  that have non-empty intersection with  $\mathcal{S}$ , that is,  $\mathcal{C} := \{\mathcal{G} + \beta \mid \beta \in \mathbb{F}_2^n, (\mathcal{G} + \beta) \cap \mathcal{S} \neq \emptyset\}$ . Define the bucket complexity of  $f$  with respect to  $\mathcal{G}$ , denoted  $\mathcal{B}(f, \mathcal{G})$ , as

$$\mathcal{B}(f, \mathcal{G}) = |\mathcal{C}|.$$

We now make the following useful observation, which follows from Equation (6).

► **Observation 8.** Let  $\Gamma$  and  $\mathcal{G}$  be as in Definition 7. Let  $\mathbf{b} = (b_1, \dots, b_t) \in \mathbb{F}_2^t$  be arbitrary. Let  $V$  be the affine subspace  $\{x \in \mathbb{F}_2^n \mid \gamma_1(x) = b_1, \dots, \gamma_t(x) = b_t\}$ . Let  $k'$  be the Fourier sparsity of  $f|_V$ . Then  $k' \leq \mathcal{B}(f, \mathcal{G})$ .

► **Definition 9** (Identification of characters). For  $f, \mathcal{G}$ , and  $\mathcal{C}$  as in Definition 7 and any  $\beta, \delta \in \mathcal{S}$ , we say that  $\beta$  and  $\delta$  are identified with respect to  $\mathcal{G}$  if  $\beta + \delta \in \mathcal{G}$ , or equivalently, if  $\beta$  and  $\delta$  belong to the same coset in  $\mathcal{C}$ .

The following observation plays a key role in the results discussed in this paper.

► **Observation 10.** Let  $f, \mathcal{G}$  and  $\mathcal{C}$  be as in Definition 7. If there exists a set  $L \subseteq \mathcal{S}$  of size  $h$  such that each  $\beta \in L$  is identified with some other  $\delta \in \mathcal{S}$  with respect to  $\mathcal{G}$ , then  $\mathcal{B}(f, \mathcal{G}) \leq k - \frac{h}{2}$ .

**Proof.** Since  $|\bar{L}| = k - h$ , there are at most  $k - h$  cosets in  $\mathcal{C}$  that contain at least one element from  $\bar{L}$ . Next, each coset in  $\mathcal{C}$  that contains only elements from  $L$  has at least 2 elements (by the hypothesis). Hence, the number of cosets containing only elements from  $L$  is at most  $h/2$ . Combining the above two, we have that  $|\mathcal{C}| \leq (k - h) + \frac{h}{2} = k - \frac{h}{2}$ . ◀

## 2.2 Folding properties of Boolean functions

► **Definition 11.** Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. We say that  $f$  is  $(\delta, \ell)$ -folding if

$$\left| \left\{ (\alpha, \beta) \in \binom{\mathcal{S}}{2} \mid |D_{\alpha+\beta}| \geq k^\ell + 1 \right\} \right| \geq \delta \binom{k}{2}.$$

Proposition 5 implies that any Boolean function is  $(1, 0)$ -folding.

We next show by a simple averaging argument that if  $f$  has “good folding properties”, then there are many  $\alpha \in \mathcal{S}$ , such that  $|D_{\alpha+\beta}|$  is large for many  $\beta \in \mathcal{S} \setminus \{\alpha\}$ .

▷ **Claim 12.** Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be  $(\delta, \ell)$ -folding and  $k \geq 6$ . Define

$$U := \left\{ \alpha \in \mathcal{S} \mid \text{there exist at least } \delta k/2 \text{ many } \beta \in \mathcal{S} \setminus \{\alpha\} \text{ with } |D_{\alpha+\beta}| \geq k^\ell + 1 \right\}.$$

Then  $|U| \geq \frac{\delta k}{3}$ .

**Proof.** For each  $\alpha \in \mathcal{S}$ , define  $t(\alpha) := |\{\beta \in \mathcal{S} \setminus \{\alpha\} \mid |D_{\alpha+\beta}| \geq k^\ell + 1\}|$ . By the hypothesis,  $\sum_{\alpha \in \mathcal{S}} t(\alpha) \geq \delta k(k-1)$ . We have

$$\begin{aligned} |U| \cdot k + (k - |U|) \cdot \frac{\delta k}{2} &\geq \sum_{\alpha \in \mathcal{S}} t(\alpha) \geq \delta k(k-1) \\ \implies |U| \left( k - \frac{\delta k}{2} \right) &\geq \delta k^2 - \delta k - \frac{\delta k^2}{2} \implies |U| \geq \frac{\delta(k-2)}{2-\delta}, \end{aligned}$$

implying  $|U| \geq \frac{\delta k}{3}$  for  $k \geq 6$ . ◀

### 3 Our contributions

In this section we give a high-level account of our contributions in this paper. In Section 3.1 we discuss the PDT construction of Tsang et al. We motivate, state our results, and briefly discuss proof ideas in Sections 3.2, 3.3, and 3.4.

#### 3.1 Low bucket complexity implies shallow PDTs

The following lemma follows from [14, Lemma 28] and Equation (4).

► **Lemma 13** (Tsang, Wong, Xie, and Zhang). *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. Then there exists an affine subspace  $V$  of  $\mathbb{F}_2^n$  of co-dimension  $O(\sqrt{k})$  such that  $f$  is constant on  $V$ .*

Let  $V = \{x \in \mathbb{F}_2^n \mid \gamma_1(x) = b_1, \dots, \gamma_t(x) = b_t\}$  be the affine subspace  $V$  obtained from Lemma 13, where  $t = O(\sqrt{k})$ . Define  $\mathcal{G} := \text{span}\{\gamma_1, \dots, \gamma_t\}$ . We next observe that  $\mathcal{B}(f, \mathcal{G}) \leq k/2$ . To see this, note that since  $f|_V$  is constant, we have from Equation (6) that for each coset  $C \in \mathcal{C}$  and any  $(b_1, \dots, b_t) \in \mathbb{F}_2^t$ ,

$$P_C(b_1, \dots, b_t) = \begin{cases} \pm 1 & \text{if } 0^n \in C \\ 0 & \text{otherwise.} \end{cases}$$

Since  $f$  is a non-constant function, this implies that each  $P_C(\cdot)$  has at least 2 terms, i.e., each  $\beta \in \mathcal{S}$  is identified with some other  $\delta \in \mathcal{S}$  with respect to  $\mathcal{G}$ . Observation 10 implies that  $\mathcal{B}(f, \mathcal{G}) \leq k/2$ . Observation 8 implies that the Fourier sparsity of the restriction of  $f$  to each coset of  $V$  is at most  $k/2$ .

This immediately leads to a recursive construction of a PDT for  $f$  of depth  $O(\sqrt{k})$  as follows. The first step is to query the parities  $\gamma_1, \dots, \gamma_t$ . After this step, each leaf of the partial tree obtained is a restriction of  $f$  to some coset of  $V$ . Next we recursively compute each leaf. Since after each batch of queries, the sparsity reduces by a factor of 2, the depth of the tree thus obtained is  $O\left(\sqrt{k} + \sqrt{\frac{k}{2}} + \sqrt{\frac{k}{2^2}} + \dots\right) = O(\sqrt{k})$ .

#### 3.2 A random set of parities achieves low bucket complexity

Tsang et al. proved Lemma 13 by an iterative procedure in each step of which a single parity is carefully chosen. We show in this paper that a randomly sampled set of parities achieves the desired bucket complexity upper bound with high probability. More specifically, for a parameter  $p \in [0, 1]$ , consider the procedure  $\text{SAMPLEPARITY}(f, p)$  described in Algorithm 1. Our first result shows that the set  $\mathcal{R}$  returned by  $\text{SAMPLEPARITY}\left(f, \frac{1}{\Theta(\sqrt{k})}\right)$

##### Algorithm 1

---

```

procedure SAMPLEPARITY ( $f, p$ )
   $\mathcal{R} \leftarrow \emptyset$ ;
  for each  $\alpha \in \mathcal{S}$  do
    independently with probability  $p$ ,  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\alpha\}$ ;
  end for
  Return  $\mathcal{R}$ ;
end procedure

```

---

satisfies  $\mathcal{B}(f, \text{span } \mathcal{R}) \leq (1 - \Omega(1))k$  with high probability.

## 29:8 On Parity Decision Trees for Fourier-Sparse Boolean Functions

► **Theorem 14.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be a Boolean function and  $k$  be sufficiently large. Let  $p = \frac{1}{2k^{1/2}}$  and  $\mathcal{R}$  be the random set of parities returned by  $\text{SAMPLEPARITY}(f, p)$ . There exists a constant  $c \in [0, 1)$  such that*

$$\mathbb{E}[\mathcal{B}(f, \text{span } \mathcal{R})] \leq ck.$$

With high probability we have  $|\mathcal{R}| = O(\sqrt{k})$ . By an argument analogous to the discussion in the previous section, Theorem 14 recovers the  $O(\sqrt{k})$  upper bound on  $\text{PDT}(f)$ . An additional insight that our work provides is that a PDT of depth  $O(\sqrt{k})$  can be obtained by a naive sampling procedure applied iteratively.

We note here that while Tsang et al. prove a bucket complexity upper bound of  $k/2$  via Lemma 13 which restricts the function to a constant, we derive a bucket complexity upper bound of  $(1 - \Omega(1))k$  by analyzing the procedure  $\text{SAMPLEPARITY}$ .

### Proof idea

Fix any  $\alpha \in \mathcal{S}$ . Proposition 5 implies that for every  $\beta \in \mathcal{S} \setminus \{\alpha\}$ , there exists  $(\gamma, \delta) \in \binom{\mathcal{S}}{2} \setminus \{(\alpha, \beta)\}$  such that  $\alpha + \beta = \gamma + \delta$ . Observe that if two parities in the set  $A := \{\beta, \gamma, \delta\}$  are chosen in  $\mathcal{R}$ , then  $\alpha$  is identified with the third parity in  $A$  w.r.t.  $\text{span } \mathcal{R}$ . Now, the expected number of  $\beta \in \mathcal{S} \setminus \{\alpha\}$  for which the aforementioned identification occurs is seen by linearity of expectation to be  $\Omega(kp^2)$ , which is  $\Omega(1)$  by the choice of  $p$ . The crux of the proof is in strengthening this bound on expectation to conclude that with constant probability, there exists at least one  $\beta \in \mathcal{S} \setminus \{\alpha\}$  such that the above identification occurs. Theorem 14 follows by linearity of expectation over  $\alpha \in \mathcal{S}$ , and an invocation of Observation 10.

We prove Theorem 14 in Section 4.2. In Section 4.1 we prove a weaker statement that admits a simpler proof, and yet contains some key ideas that go into the proof of Theorem 14.

### 3.3 Good folding yields better PDTs

Assume that for any Boolean function  $f$  there exist  $\alpha_1, \alpha_2 \in \mathcal{S}$  such that  $|D_{\alpha_1 + \alpha_2}| \geq k^\ell + 1$ . This is a weaker assumption on  $f$  than it being  $(\delta, \ell)$ -folding. Observation 10 implies that  $\mathcal{B}(f, \{0^n, \alpha_1 + \alpha_2\}) \leq k - k^\ell - 1 \leq k(1 - k^{-(1-\ell)})$ . This suggests the following PDT for  $f$ . First the parity  $\alpha_1 + \alpha_2$  is queried at the root. Observation 8 implies that the Fourier sparsity of  $f$  restricted to the affine subspace (of co-dimension 1) corresponding to each outcome of this query is at most  $k(1 - k^{-(1-\ell)})$ . Repeating this heuristic recursively for each leaf leads to a PDT of depth  $O(k^{1-\ell} \log k)$ .

We have now set up the backdrop to introduce our next contribution. In the preceding discussion we had assumed the following about any Boolean function  $f$ : there exists a pair in  $\binom{\mathcal{S}}{2}$  with a large equivalence class. One implication of our next result is that if we instead assume that any Boolean function is  $(\Omega(1), \ell)$ -folding, the procedure  $\text{SAMPLEPARITY}$  with  $p$  set to  $1/\tilde{\Theta}(k^{(1+\ell)/2})$  achieves a bucket complexity upper bound of  $k(1 - \Omega(1))$  with high probability. By an argument analogous to the discussion in Section 3.1 (also see Corollary 16), this yields a PDT with depth  $\tilde{O}(k^{(1-\ell)/2})$ . This is a quadratic improvement over the  $\tilde{O}(k^{1-\ell})$  bound discussed in the last paragraph. Besides, it can be seen to recover (up to a logarithmic factor) our first result by setting  $\ell = 0$ , since any Boolean function is  $(1, 0)$ -folding by Proposition 5.

► **Theorem 15.** *Let  $0 \leq \ell \leq 1 - \Omega(1)$  and  $\delta \in (0, 1]$ . Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be  $(\delta, \ell)$ -folding with  $k$  sufficiently large. Set  $p := \frac{4000 \log k}{\delta k^{(1+\ell)/2}}$  and let  $\mathcal{R}$  be the random subset of  $\mathcal{S}$  that  $\text{SAMPLEPARITY}(f, p)$  returns. Then with probability at least  $1 - \frac{1}{k}$ ,  $\mathcal{B}(f, \text{span } \mathcal{R}) \leq k - \frac{\delta k}{6}$ .*

The proof of Theorem 15 proceeds along the lines of that of Theorem 14, but is more technical. A proof of it can be found in Section 5 of the full version of our paper [8].

This yields the following corollary.

► **Corollary 16.** *Let  $0 \leq \ell \leq 1 - \Omega(1)$  and  $\delta = \Omega(1)$ . Suppose all Boolean functions  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  with sufficiently large  $k$  are  $(\delta, \ell)$ -folding. Then,*

$$\text{PDT}(f) = \tilde{O}(k^{(1-\ell)/2}).$$

**Proof.** Fix any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  with sufficiently large  $k$ . Let  $p$  and  $\mathcal{R}$  be as in the statement of Theorem 15. Since  $\delta$  is a constant,  $p = \Theta\left(\frac{\log k}{k^{(1+\ell)/2}}\right)$ . By Theorem 15, we have  $\mathcal{B}(f, \text{span } \mathcal{R}) \leq ck$ , for some  $c = (1 - \Omega(1))$ , with probability strictly greater than  $1/2$ . By a Chernoff bound  $|\mathcal{R}| = \tilde{O}(k^{(1-\ell)/2})$  with probability strictly greater than  $1/2$ . Finally, by a union bound, we have that with non-zero probability the set  $\mathcal{R}$  returned by  $\text{SAMPLEPARITY}(f, p)$  satisfies both  $|\mathcal{R}| = \tilde{O}(k^{(1-\ell)/2})$  and  $\mathcal{B}(f, \text{span } \mathcal{R}) \leq ck$ , for some  $c = (1 - \Omega(1))$ . Choose such an  $\mathcal{R}$  and consider the following PDT for  $f$ , whose construction closely follows the discussion in Section 3.1.

First, query all parities in  $\mathcal{R}$ . Now, let  $V$  be the affine subspace corresponding to an arbitrary leaf of this partial tree. By the properties of  $\mathcal{R}$  and Observation 8, we have that the Fourier sparsity of  $f|_V$  is at most  $ck$ . Repeat the same process inductively for each leaf. The depth of the resultant tree is at most  $\tilde{O}(k^{(1-\ell)/2} + (ck)^{(1-\ell)/2} + \dots) = \tilde{O}(k^{(1-\ell)/2})$ . ◀

Corollary 16 naturally raises the question of whether all Boolean functions are  $(\Omega(1), \Omega(1))$ -folding.

► **Question 17.** *Do there exist constants  $\ell, \delta \in (0, 1]$  such that every Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  is  $(\delta, \ell)$ -folding?*

An affirmative answer to Question 17 in conjunction with Corollary 16 and the discussion in Section 1 implies an upper bound on the communication complexity of XOR functions  $F = f \circ \oplus$  that is polynomially smaller than the best known bound of  $O(\sqrt{\text{rank}(F)})$ .

What is the largest  $\ell$  for which all Boolean functions are  $(\Omega(1), \ell)$ -folding? The addressing function  $\text{ADD}_k$  (see Definition 3) is  $(1, 1/2 - o(1))$ -folding, and not  $(\Omega(1), \ell)$ -folding for any  $\ell \geq \frac{1}{2}$  (see [8, Appendix B]). In light of this, we make the following conjecture.

► **Conjecture 18.** *There exists a constant  $\delta > 0$  such that any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  is  $(\delta, 1/2 - o(1))$ -folding.*

Assuming Conjecture 18, Corollary 16 would imply an upper bound of  $\tilde{O}(\text{rank}^{1/4+o(1)}(F))$  on the communication complexity of XOR functions  $F = f \circ \oplus$ .

### 3.4 Boolean functions have non-trivial folding properties

Recall that Conjecture 18 states that any Boolean function is  $(\delta, \ell)$ -folding with  $\delta = \Omega(1)$  and  $\ell = 1/2 - o(1)$ . Also recall from Proposition 5 that a necessary condition for a function to be Boolean valued is that it is  $(\delta, \ell)$ -folding with  $\delta = 1$  and  $\ell = 0$ . We show in the Section 6 (see Theorem 27) that the conditions in Proposition 5 are not sufficient for a function to be Boolean valued.

To the best of our knowledge, it was not known prior to our work whether any better bound than this was known for Boolean functions (in terms of  $\ell$ , for *any* non-zero  $\delta$ ). In particular, it was consistent with prior knowledge that there exist functions for which each equivalence class of  $\binom{S}{2}$  contains exactly 2 elements. We rule out this possibility, and our contribution is a step towards Conjecture 18.

## 29:10 On Parity Decision Trees for Fourier-Sparse Boolean Functions

► **Theorem 19.** *For any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  with  $k > 4$ , and every  $\alpha \in \mathcal{S}$ , there exists  $\beta \in \mathcal{S} \setminus \{\alpha\}$  such that  $|D_{\alpha+\beta}| \geq 3$ .*

In order to rule out the possibility mentioned above, it suffices to exhibit a single pair  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$  with  $|D_{\alpha+\beta}| \geq 3$ . Theorem 19 further shows that *every* element  $\alpha \in \mathcal{S}$  participates in such a pair.

### Proof idea

We prove this via a series of arguments. Define  $\mathcal{S}_+ := \{\alpha \in \mathcal{S} \mid \widehat{f}(\alpha) > 0\}$  and  $\mathcal{S}_- := \{\alpha \in \mathcal{S} \mid \widehat{f}(\alpha) < 0\}$ . We first show that if there exists  $\alpha \in \mathcal{S}$  with  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ , then both of the following hold.

1. Either  $|\mathcal{S}_+|$  or  $|\mathcal{S}_-|$  is odd.
2. The function  $f$  must be plateaued.

The proofs use Equation (3). Next, we show that for plateaued Boolean functions, both  $|\mathcal{S}_+|$  and  $|\mathcal{S}_-|$  are even, yielding a contradiction in view of the first bullet above. This proof involves a careful analysis of the Fourier coefficients and crucially uses Observation 1 and Equation (2).

A natural question raised by Theorem 19 is whether there exists a Boolean function  $f$  and  $\alpha \in \mathcal{S}$  such that there exists only one element  $\beta \in \mathcal{S} \setminus \{\alpha\}$  with  $|D_{\alpha+\beta}| \geq 3$ . The following theorem answers this question in the positive, and sheds more light on the structure of such functions.

► **Theorem 20.**

1. *There exists a Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  and  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$  such that  $|D_{\alpha+\gamma}| = 2$  for all  $\gamma \in \mathcal{S} \setminus \{\alpha, \beta\}$ .*
2. *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. If there exists  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$  such that  $|D_{\alpha+\gamma}| = 2$  for all  $\gamma \in \mathcal{S} \setminus \{\alpha, \beta\}$ , then  $|D_{\alpha+\beta}| = k/2$ .*

The proof of Part 2 of Theorem 20 follows along the lines of the proof of Theorem 19. The proof of Part 1 of Theorem 20 constructs such a function by considering any Boolean function and applying a simple modification to it.

We prove Theorem 19 in Section 5 and Theorem 20 in [8, Section 6].

## 4 Proof of Theorem 14

In this Section, we prove our first result, Theorem 14.

### 4.1 Warm up: sampling $\widetilde{O}(k^{3/4})$ parities.

In this section we prove a quantitatively weaker statement. This admits a simpler proof and introduces many key ideas that go into our proof of Theorem 14.

▷ **Claim 21.** Let  $p := \frac{2\sqrt{\log k}}{k^{1/4}}$ , and let  $\mathcal{R}$  be the set returned by  $\text{SAMPLEPARITY}(f, p)$ . Then

$$\Pr[\mathcal{B}(f, \text{span } \mathcal{R}) \leq k/2] \geq 1 - \frac{1}{k^{1/3}}.$$

By a Chernoff bound, with high probability,  $|\mathcal{R}| = \widetilde{O}(k^{3/4})$ .

Proof. Fix any  $\alpha \in \mathcal{S}$ . By Proposition 5 we have that for each  $\beta \in \mathcal{S} \setminus \{\alpha\}$ , there exist  $\beta_1, \beta_2 \in \mathcal{S} \setminus \{\alpha, \beta\}$  such that  $\alpha + \beta + \beta_1 + \beta_2 = 0$ . Fix any such  $\beta_1, \beta_2$ , and define  $Q_\beta := \{\beta, \beta_1, \beta_2\}$ . Note that the sets  $Q_\beta$  are not necessarily distinct. Define the multiset of unordered triples  $\mathcal{F} := \{Q_\beta \mid \beta \in \mathcal{S} \setminus \{\alpha\}\}$ . For each  $\gamma \in \mathcal{S} \setminus \{\alpha\}$ , define  $\mathcal{D}_\gamma := \{\beta \in \mathcal{S} \setminus \{\alpha\} \mid \gamma \in Q_\beta\}$ . We now show that with high probability there exists  $F \in \mathcal{F}$  such that  $|F \cap \mathcal{R}| \geq 2$ . We consider two cases below.

**Case 1: There exists  $\gamma \in \mathcal{S} \setminus \{\alpha\}$  such that  $|\mathcal{D}_\gamma| \geq k^{1/2}$ .**

Consider the multiset of unordered pairs  $\mathcal{A} := \{Q_\beta \setminus \{\gamma\} \mid \beta \in \mathcal{D}_\gamma\}$ . Each pair in  $\mathcal{A}$  can repeat at most thrice. Hence there are at least  $k^{1/2}/3$  distinct pairs in  $\mathcal{A}$ . Moreover the distinct pairs in  $\mathcal{A}$  are disjoint. This can be inferred from the observation that the sum of the two elements in each pair in  $\mathcal{A}$  equals  $\alpha + \gamma$ . Thus

$$\Pr[\forall A \in \mathcal{A}, A \not\subseteq \mathcal{R}] \leq (1 - p^2)^{k^{1/2}/3} = \left(1 - \frac{4 \log k}{k^{1/2}}\right)^{k^{1/2}/3} \leq \frac{1}{k^{4/3}}.$$

**Case 2: For each  $\gamma \in \mathcal{S} \setminus \{\alpha\}$ ,  $|\mathcal{D}_\gamma| < k^{1/2}$ .**

In this case each triple in  $\mathcal{F}$  has non-empty intersection with at most  $3k^{1/2}$  sets in  $\mathcal{F}$ . Thus one can greedily obtain a collection  $\mathcal{T}$  of at least  $\frac{k-1}{3k^{1/2}}$  disjoint triples in  $\mathcal{F}$ .

$$\Pr[\forall T \in \mathcal{T}, |T \cap \mathcal{R}| < 2] \leq (1 - p^2)^{\frac{k-1}{3k^{1/2}}} = \left(1 - \frac{4 \log k}{k^{1/2}}\right)^{\frac{k-1}{3k^{1/2}}},$$

which is at most  $\frac{1}{k^{4/3}}$  for sufficiently large  $k$ .

From the above two cases it follows that with probability at least  $1 - \frac{1}{k^{4/3}}$ , there exists a triple  $F \in \mathcal{F}$  such that  $|F \cap \mathcal{R}| \geq 2$ . Assume existence of such a triple  $F$ , and let  $\delta_1, \delta_2 \in F \cap \mathcal{R}$ . Let  $\delta := F \setminus \{\delta_1, \delta_2\}$ . Since  $\alpha + \delta_1 + \delta_2 + \delta = 0^n$ , we have that  $\alpha + \delta = \delta_1 + \delta_2 \in \text{span } \mathcal{R}$ , i.e.,  $\alpha$  is identified with  $\delta$  with respect to  $\text{span } \mathcal{R}$ . By a union bound over all  $\alpha \in \mathcal{S}$  it follows that with probability at least  $1 - \frac{1}{k^{1/3}}$ , for every  $\alpha \in \mathcal{S}$  there exists a  $\delta \in \mathcal{S} \setminus \{\alpha\}$  such that  $\alpha$  is identified with  $\delta$  w.r.t.  $\mathcal{R}$ . The claim follows by Observation 10.  $\triangleleft$

## 4.2 Sampling $O(k^{1/2})$ parities

We now proceed to prove Theorem 14 by refining the ideas developed in Section 4.1. Recall that by a Chernoff bound,  $|\mathcal{R}| = O(\sqrt{k})$  with high probability (where  $\mathcal{R}$  is as in Theorem 14). We require the following inequality whose proof can be found in [8, Section 4.2].

► **Proposition 22.** *For any non-negative integer  $d$ , and  $p \in [0, 1]$  be such that  $pd \leq 1$ . Then,*

$$(1 - p)^d \leq 1 - \frac{1}{2}pd.$$

**Proof of Theorem 14.** For technical reasons we instead consider a two-step probabilistic procedure. Define  $p' := \frac{1}{4k^{1/2}}$ . Let  $\mathcal{R}_1$  and  $\mathcal{R}_2$  be the sets returned by two independent runs of  $\text{SAMPLEPARITY}(f, p')$ , and let  $\mathcal{R}' := \mathcal{R}_1 \cup \mathcal{R}_2$ . Each  $\alpha \in \mathcal{S}$  is independently included in  $\mathcal{R}'$  with probability equal to  $1 - (1 - p')^2 < 2p' = p$ . Hence it suffices to prove that there exists a constant  $c \in (0, 1]$  such that  $\mathbb{E}[\mathcal{B}(f, \text{span } \mathcal{R}')] \leq ck$ .

Fix any  $\alpha \in \mathcal{S}$  and let  $Q_\beta$  and  $\mathcal{F}$  be as in the proof of Claim 21. For  $\gamma \in \mathcal{S} \setminus \{\alpha\}$ , define  $\widetilde{\text{deg}}(\gamma) := |\{\beta \in \mathcal{S} \setminus \{\alpha\} \mid \gamma \in Q_\beta \setminus \{\beta\}\}|$ . Clearly,  $\mathbb{E}_{\gamma \sim \mathcal{S} \setminus \{\alpha\}}[\widetilde{\text{deg}}(\gamma)] = 2$ . Define  $A := \{\gamma \in \mathcal{S} \setminus \{\alpha\} \mid \widetilde{\text{deg}}(\gamma) \geq 4k^{1/2}\}$ . By Markov's inequality,  $|A| \leq k^{1/2}/2$ . Fix an ordering  $\sigma$  on  $\mathcal{S} \setminus \{\alpha\}$  such that all elements of  $\bar{A} := (\mathcal{S} \setminus \{\alpha\}) \setminus A$  appear before all elements of  $A$ .

## 29:12 On Parity Decision Trees for Fourier-Sparse Boolean Functions

Define  $T := \{\beta \in \mathcal{S} \setminus \{\alpha\} \mid Q_\beta \setminus \{\beta\} \subseteq A\}$ . Observe that the pairs  $Q_\beta \setminus \{\beta\}$  for distinct  $\beta \in \mathcal{S} \setminus \{\alpha\}$  are distinct. This can be inferred from the observation that the sum (with respect to coordinate-wise addition in  $\mathbb{F}_2$ ) of the two elements of  $Q_\beta \setminus \{\beta\}$  equals  $\alpha + \beta$ . This gives us the following bound on the size of  $T$ :

$$|T| \leq \binom{|A|}{2} \leq \frac{k}{8}. \quad (7)$$

Define  $\bar{T} := (\mathcal{S} \setminus \{\alpha\}) \setminus T$ . For each  $\beta \in \bar{T}$ , the first character (according to  $\sigma$ ) in the pair  $Q_\beta \setminus \{\beta\}$  is from  $\bar{A}$ . For each  $\gamma \in \bar{A}$ , define  $d(\gamma)$  to be the number of  $\beta \in \bar{T}$  such that  $\gamma$  is the first element in  $Q_\beta \setminus \{\beta\}$ . By Equation (7) we have

$$\sum_{\gamma \in \bar{A}} d(\gamma) = |\bar{T}| \geq k - 1 - \frac{k}{8} \geq \frac{2k}{3} \quad (8)$$

where the last inequality holds for sufficiently large  $k$ .

For  $\gamma \in \bar{A}$ , let  $\mathcal{E}(\gamma)$  be the event that there exists  $\beta \in \bar{T} \cap \mathcal{R}_1$  such that  $\gamma$  is the first element in  $Q_\beta \setminus \{\beta\}$ . We have

$$\Pr_{\mathcal{R}_1}[\mathcal{E}(\gamma)] = 1 - (1 - p')^{d(\gamma)} \geq \frac{p' \cdot d(\gamma)}{2}, \quad (9)$$

where the last inequality follows by Proposition 22. Here Proposition 22 is applicable since  $d(\gamma) \leq \widehat{\deg}(\gamma) \leq 4k^{1/2}$  (since  $\gamma \in \bar{A}$ ), and  $p' = \frac{1}{4k^{1/2}}$ . Define the random set  $B := \{\gamma \in \bar{A} \mid \mathcal{E}(\gamma) \text{ occurs}\}$ . We have

$$\begin{aligned} \mathbb{E}_{\mathcal{R}_1}[|B|] &= \sum_{\gamma \in \bar{A}} \Pr_{\mathcal{R}_1}[\mathcal{E}(\gamma)] \geq \sum_{\gamma \in \bar{A}} \frac{p' \cdot d(\gamma)}{2} \quad \text{by linearity of expectation and Equation (9)} \\ &\geq \frac{1}{2} \cdot \frac{1}{4k^{1/2}} \cdot \frac{2k}{3} \geq \frac{k^{1/2}}{12}. \quad \text{by Equation (8), and substituting the value of } p' \end{aligned}$$

Furthermore, the events  $\mathcal{E}(\gamma)$  are independent. By a Chernoff bound,  $\Pr_{\mathcal{R}_1}[|B| \geq \frac{k^{1/2}}{24}] \geq 0.9$ . Now,

$$\begin{aligned} \Pr_{\mathcal{R}_1, \mathcal{R}_2} \left[ B \cap \mathcal{R}_2 \neq \emptyset \mid |B| \geq \frac{k^{1/2}}{24} \right] &\geq 1 - (1 - p')^{k^{1/2}/24} \geq 1 - e^{-p' \cdot \frac{k^{1/2}}{24}} = 1 - e^{-\frac{1}{96}} \\ &= c_1, \text{ say.} \end{aligned}$$

Thus, the probability of the event  $\mathcal{E} := \left\{ |B| \geq \frac{k^{1/2}}{24} \right\} \wedge \{B \cap \mathcal{R}_2 \neq \emptyset\}$  is at least  $0.9c_1$ . Suppose the event  $\mathcal{E}$  occurs, and let  $\gamma \in B \cap \mathcal{R}_2$ . By the definitions of  $B$  and  $\mathcal{E}(\gamma)$ , there exists  $\beta \in \bar{T} \cap \mathcal{R}_1$  such that  $\gamma$  is the first element of  $Q_\beta \setminus \{\beta\}$ . Let  $\delta := Q_\beta \setminus \{\beta, \gamma\}$ . Then,  $\alpha + \delta = \beta + \gamma$ . Since  $\beta \in \mathcal{R}_1$  and  $\gamma \in \mathcal{R}_2$ ,  $\alpha$  is identified with  $\delta$  with respect to  $\text{span } \mathcal{R}'$ . In summary, we have shown that for any  $\alpha \in \mathcal{S}$ ,

$$\Pr_{\mathcal{R}_1, \mathcal{R}_2} [\alpha \text{ is identified with some } \delta \in \mathcal{S} \setminus \{\alpha\} \text{ w.r.t. } \text{span } \mathcal{R}'] \geq 0.9c_1.$$

By linearity of expectation,

$$\mathbb{E}_{\mathcal{R}_1, \mathcal{R}_2} [|\{\alpha \in \mathcal{S} \mid \alpha \text{ is identified with some } \delta \in \mathcal{S} \setminus \{\alpha\} \text{ w.r.t. } \text{span } \mathcal{R}'\}|] \geq k \cdot 0.9c_1.$$

Observation 10 then implies

$$\mathbb{E}_{\mathcal{R}_1, \mathcal{R}_2} [\mathcal{B}(f, \text{span } \mathcal{R}')] \leq k - \frac{k \cdot 0.9c_1}{2} = ck,$$

where  $c = \left(1 - \frac{0.9c_1}{2}\right)$ . ◀

## 5 Proof of Theorem 19

In this section we prove Theorem 19, which states that for any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  and  $\alpha \in \mathcal{S}$ , there exists at least one  $\beta \in \mathcal{S}$  with  $|D_{\alpha+\beta}| \geq 3$ .

We first recall and introduce some notation. Recall from Proposition 5 that for any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  and every  $\gamma \in (\mathcal{S} + \mathcal{S}) \setminus \{0^n\}$ , we have  $|D_\gamma| \geq 2$ . For any  $\gamma$  with  $|D_\gamma| > 2$ , we say that  $\gamma$  is a *non-trivial folding direction*. Hence, Theorem 19 can be rephrased to say that for any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ , every element  $\alpha \in \mathcal{S}$  must participate in at least one non-trivial folding direction. For any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ , define  $\mathcal{S}_+ := \{\alpha \in \mathcal{S} \mid \widehat{f}(\alpha) > 0\}$ , and  $\mathcal{S}_- := \{\alpha \in \mathcal{S} \mid \widehat{f}(\alpha) < 0\}$ . For any set  $S$ , we use the notation  $\binom{S}{3}$  to denote the set of all subsets of  $S$  of size exactly 3. We abuse notation and denote a generic element of  $\binom{S}{3}$  as  $(a, b, c)$  rather than  $\{a, b, c\}$ .

We require the following proposition. For a proof, refer to [8, Section 6].

► **Proposition 23.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be a Boolean function with Fourier support  $\mathcal{S}$  with  $k = |\mathcal{S}| \geq 2$ . Let  $\alpha, \beta$  be two distinct parities in  $\mathcal{S}$ . Then, there exists a Boolean function  $g : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  with Fourier support  $\mathcal{S}$  and  $\widehat{g}(\alpha) > 0, \widehat{g}(\beta) > 0$ .*

We next state a preliminary claim.

▷ **Claim 24.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. Suppose there exists  $\alpha \in \mathcal{S}$  such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ . Then, either  $|\mathcal{S}_+|$  is odd or  $|\mathcal{S}_-|$  is odd.*

*Proof.* Fix any set  $\alpha \in \mathcal{S}$  such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ . Assume  $\alpha \in \mathcal{S}_+$  (else run this argument with  $\mathcal{S}_+$  and  $\mathcal{S}_-$  interchanged). Consider the set of unordered triples

$$T = \left\{ (\beta, \gamma, \delta) \in \binom{\mathcal{S} \setminus \{\alpha\}}{3} \mid \alpha + \beta + \gamma + \delta = 0^n \right\}.$$

Let  $T_+$  denote the set of triples in  $T$  that contain at least one element  $\beta \in \mathcal{S}_+$ , i.e.,

$$T_+ := \left\{ (\beta, \gamma, \delta) \in T \mid \text{at least one of } \widehat{f}(\beta), \widehat{f}(\gamma), \widehat{f}(\delta) \text{ is positive} \right\}.$$

Since  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ , this implies that any  $\beta \in \mathcal{S}$  (in particular any  $\beta \in \mathcal{S}_+$ ) appears in exactly one triple. For any  $\beta \in \mathcal{S}_+$ , say this triple is  $(\beta, \beta_1, \beta_2)$ . Equation (3) implies that

$$\widehat{f}(\alpha)\widehat{f}(\beta) + \widehat{f}(\beta_1)\widehat{f}(\beta_2) = 0.$$

Since  $\alpha$  and  $\beta$  are both in  $\mathcal{S}_+$ , exactly one of  $\beta_1, \beta_2$  is in  $\mathcal{S}_+$  and the other is in  $\mathcal{S}_-$ .

Thus each triple in  $T_+$  contains exactly two elements of  $\mathcal{S}_+$ , and none of these elements appears in any other triple. Moreover each element of  $\mathcal{S}_+$  appears in some triple in  $T_+$ . Accounting for  $\alpha$  being in  $\mathcal{S}_+$ , we conclude that if  $|T_+| = t$ , then  $|\mathcal{S}_+| = 2t + 1$ , which is odd. ◁

We state another claim that we require.

▷ **Claim 25.** *Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any Boolean function. If there exists  $\alpha \in \mathcal{S}$  such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ , then  $f$  is plateaued.*

*Proof.* Fix any  $\alpha \in \mathcal{S}$  such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ . Towards a contradiction, suppose  $f$  is not plateaued. This implies existence of  $\gamma \in \mathcal{S}$  such that  $|\widehat{f}(\alpha)| \neq |\widehat{f}(\gamma)|$ . Proposition 5 implies existence of  $\mu, \nu \in \mathcal{S}$  be such that  $\alpha + \gamma = \mu + \nu$ . We also have that

$$\alpha + \nu = \mu + \gamma, \quad \alpha + \mu = \gamma + \nu.$$

## 29:14 On Parity Decision Trees for Fourier-Sparse Boolean Functions

Arrange  $\alpha, \gamma, \mu$  and  $\nu$  in non-increasing order of the absolute values of their Fourier coefficients. Let the resultant sequence be  $\delta_1, \delta_2, \delta_3, \delta_4$ . Thus,

$$|\widehat{f}(\delta_1)| \geq |\widehat{f}(\delta_2)| \geq |\widehat{f}(\delta_3)| \geq |\widehat{f}(\delta_4)|.$$

Since  $|\widehat{f}(\alpha)| \neq |\widehat{f}(\gamma)|$ , at least one of these inequalities must be strict, which in particular implies that  $|\widehat{f}(\delta_1)||\widehat{f}(\delta_2)| > |\widehat{f}(\delta_3)||\widehat{f}(\delta_4)|$ . Now by the hypothesis, for all  $1 \leq i < j \leq 4$ , and  $\{k, m\} := \{1, 2, 3, 4\} \setminus \{i, j\}$  we have that  $|D_{\delta_i + \delta_j}| = |D_{\delta_k + \delta_m}| = 2$ . Thus, by Equation (3) we have that  $\widehat{f}(\delta_1)\widehat{f}(\delta_2) = -\widehat{f}(\delta_3)\widehat{f}(\delta_4)$ , implying that  $|\widehat{f}(\delta_1)||\widehat{f}(\delta_2)| = |\widehat{f}(\delta_3)||\widehat{f}(\delta_4)|$ , which is a contradiction.  $\triangleleft$

The next claim shows that Theorem 19 holds true if  $f$  is a plateaued function.

$\triangleright$  **Claim 26.** Let  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$  be any plateaued Boolean function with  $k > 4$ . Then, for any  $\alpha \in \mathcal{S}$ , there exists  $\beta \in \mathcal{S} \setminus \{\alpha\}$  such that  $|D_{\alpha+\beta}| \geq 3$ .

*Proof.* Towards a contradiction, let  $\alpha \in \mathcal{S}$  be such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ . Let  $s = |\mathcal{S}_+|$  and  $t = |\mathcal{S}_-|$ . We now prove that  $s$  and  $t$  must both be even.

Since  $f$  is plateaued, Equation (2) implies that  $|\widehat{f}(\gamma)| = 1/\sqrt{k}$  for all  $\gamma \in \mathcal{S}$ . By Observation 1 we know that  $1/\sqrt{k} = c/2^n$  for some  $c \in \mathbb{Z}$ . This implies that  $k = 2^{2n}/c^2$ . Since  $k$  is an integer,  $c$  must be a power of 2, and hence  $k = 2^{2h}$  for some  $h > 1$  (since we assumed  $k > 4$ ).

Assume  $f(0^n) = 1$  (else run the same argument with  $f$  replaced by  $-f$ ). This implies

$$\sum_{\gamma \in \mathcal{S}_+} |\widehat{f}(\gamma)| - \sum_{\delta \in \mathcal{S}_-} |\widehat{f}(\delta)| = 1.$$

That is,  $(s - t)/\sqrt{k} = 1$ . Since  $s + t = k$ , this implies  $s = \frac{k}{2} + \frac{\sqrt{k}}{2}$  and  $t = \frac{k}{2} - \frac{\sqrt{k}}{2}$ . Since  $k = 2^{2h}$  for some  $h > 1$  (since we assumed  $k > 4$ ),  $s$  and  $t$  are both even. This is a contradiction in view of Claim 24.  $\triangleleft$

We next use Claim 25 to remove the assumption of  $f$  being plateaued in the previous claim, which proves Theorem 19.

**Proof of Theorem 19.** Towards a contradiction, suppose there exists  $\alpha \in \mathcal{S}$  such that  $|D_{\alpha+\beta}| = 2$  for all  $\beta \in \mathcal{S} \setminus \{\alpha\}$ . Claim 25 implies that  $f$  must be plateaued. Next, Claim 26 implies that there must exist  $\gamma \in \mathcal{S}$  such that  $|D_{\alpha+\gamma}| \geq 3$ , which is a contradiction.  $\blacktriangleleft$

## 6 Ruling out sufficiency of Proposition 5

In this section, we prove that the conditions in Proposition 5 are not sufficient for a function to be Boolean. To the best of our knowledge, ours is the first work to show this.

$\blacktriangleright$  **Theorem 27.** *There exists a set  $\mathcal{S} \subseteq \mathbb{F}_2^n$  such that  $|D_{\alpha+\beta}| \geq 2$  for all  $(\alpha, \beta) \in \binom{\mathcal{S}}{2}$ , but  $\mathcal{S}$  is not the Fourier support of any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ .*

For sets  $A, B \subseteq [n]$ , let  $A \Delta B$  denote the symmetric difference of the sets  $A$  and  $B$ . For  $x \in \mathbb{R} \setminus \{0\}$ , define  $\text{sgn}(x) := -1$  if  $x < 0$ , and  $\text{sgn}(x) := 1$  if  $x > 0$ .

**Proof.** For the purpose of this proof, we require the natural equivalence between elements of  $\mathbb{F}_2^n$  and subsets of  $[n]$ . Under this equivalence, the sum of two elements in  $\mathbb{F}_2^n$  corresponds to the symmetric difference of the corresponding sets in  $[n]$ . The following is a property of symmetric difference. For any sets  $A, B, C, D \subseteq [n]$ ,

$$A \Delta B = C \Delta D \iff A \Delta C = B \Delta D. \quad (10)$$

Hence it suffices to exhibit a collection  $\mathcal{S}$  of subsets of  $[n]$  such that for all  $(S, T) \in \binom{[S]}{2}$ , there exist  $(U, V) \neq (S, T) \in \binom{[S]}{2}$  with  $S \Delta T = U \Delta V$ , and  $\mathcal{S}$  is not the Fourier support of any Boolean function  $f : \mathbb{F}_2^n \rightarrow \{-1, 1\}$ . To this end, consider the set

$$\mathcal{S} = \{\{1\}, \dots, \{n\}, \{1, 2, n\}, \dots, \{1, n-1, n\}\}.$$

Below we list out all equivalence classes of  $\binom{[S]}{2}$ . For any distinct  $i, j \in \{2, 3, \dots, n-1\}$  we have  $\{i\} \Delta \{j\} = \{1, i, n\} \Delta \{1, j, n\}$ . Thus

$$D_{\{i\} \Delta \{j\}} = \{(\{i\}, \{j\}), (\{1, i, n\}, \{1, j, n\})\} \quad \forall i, j \in \{2, 3, \dots, n-1\}. \quad (11)$$

For any  $i \in \{2, 3, \dots, n-1\}$  we have

$$\begin{aligned} \{1\} \Delta \{i\} &= \{n\} \Delta \{1, i, n\}, \\ \{n\} \Delta \{i\} &= \{1\} \Delta \{1, i, n\}. \end{aligned}$$

We also have

$$\{1\} \Delta \{n\} = \{i\} \Delta \{1, i, n\} \quad \text{for all } i \in \{2, 3, \dots, n-1\}.$$

Along with Equation (10), these establish the fact that  $|D_{\alpha+\beta}| \geq 2$  for all  $(\alpha, \beta) \in \binom{[S]}{2}$ . We now provide a proof of the fact that  $\mathcal{S}$  cannot be the Fourier support of any Boolean function. Consider the following six sets.

$$S_1 = \{2\}, S_2 = \{3\}, S_3 = \{4\}, S_4 = \{1, 2, n\}, S_5 = \{1, 3, n\}, S_6 = \{1, 4, n\}.$$

If  $\mathcal{S}$  is the support of a Boolean function, then Equation (3) holds true. Equation (11) then implies

$$\begin{aligned} \widehat{f}(S_1)\widehat{f}(S_2) + \widehat{f}(S_4)\widehat{f}(S_5) &= 0, \\ \widehat{f}(S_1)\widehat{f}(S_3) + \widehat{f}(S_4)\widehat{f}(S_6) &= 0, \\ \widehat{f}(S_2)\widehat{f}(S_3) + \widehat{f}(S_5)\widehat{f}(S_6) &= 0. \end{aligned}$$

Let  $s_i = \text{sgn}(\widehat{f}(S_i))$  for  $i \in [6]$ . Thus,

$$\begin{aligned} s_1 s_2 &= -s_4 s_5 \\ s_1 s_3 &= -s_4 s_6 \\ s_2 s_3 &= -s_5 s_6. \end{aligned}$$

Multiplying out the left hand sides and right hand sides of the above, we obtain  $1 = -1$ , which is a contradiction. Hence  $\mathcal{S}$  cannot be the support of any Boolean function.  $\blacktriangleleft$

---

**References**

---

- 1 Anurag Anshu, Naresh Goud Boddu, and Dave Touchette. Quantum log-approximate-rank conjecture is also false. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 982–994, 2019. doi:10.1109/FOCS.2019.00063.
- 2 Anna Bernasconi and Bruno Codenotti. Spectral analysis of boolean functions as a graph eigenvalue problem. *IEEE Trans. Computers*, 48(3):345–351, 1999. doi:10.1109/12.755000.
- 3 Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 42–53, 2019. doi:10.1145/3313276.3316353.
- 4 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. *SIAM J. Comput.*, 47(1):208–217, 2018. doi:10.1137/17M1136869.
- 5 Troy Lee and Adi Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–398, 2009. doi:10.1561/0400000040.
- 6 László Lovász and Michael E. Saks. Lattices, möbius functions and communication complexity. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 81–90, 1988. doi:10.1109/SFCS.1988.21924.
- 7 Shachar Lovett. Communication is bounded by root of rank. *J. ACM*, 63(1):1:1–1:9, 2016. doi:10.1145/2724704.
- 8 Nikhil S. Mande and Swagato Sanyal. On parity decision trees for Fourier-sparse Boolean functions. *CoRR*, abs/2008.00266, 2020. arXiv:2008.00266.
- 9 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions. *CoRR*, abs/0909.3392, 2009. URL: <http://arxiv.org/abs/0909.3392>.
- 10 Swagato Sanyal. Fourier sparsity and dimension. *Theory of Computing*, 15(1):1–13, 2019.
- 11 Amir Shpilka, Avishay Tal, and Ben lee Volk. On the structure of boolean functions with small spectral norm. *Comput. Complex.*, 26(1):229–273, 2017. doi:10.1007/s00037-015-0110-y.
- 12 Makrand Sinha and Ronald de Wolf. Exponential separation between quantum communication and logarithm of approximate rank. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 966–981, 2019. doi:10.1109/FOCS.2019.00062.
- 13 Robert C Titsworth. *Correlation properties of cyclic sequences*. PhD thesis, California Institute of Technology, 1962.
- 14 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 658–667, 2013. doi:10.1109/FOCS.2013.76.

# Colored Cut Games

**Nils Morawietz**

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany  
morawietz@informatik.uni-marburg.de

**Niels Grüttemeier** 

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany  
niegru@informatik.uni-marburg.de

**Christian Komusiewicz** 

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany  
komusiewicz@informatik.uni-marburg.de

**Frank Sommer** 

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany  
fsommer@informatik.uni-marburg.de

---

## Abstract

In a graph  $G = (V, E)$  with an edge coloring  $\ell : E \rightarrow C$  and two distinguished vertices  $s$  and  $t$ , a colored  $(s, t)$ -cut is a set  $\tilde{C} \subseteq C$  such that deleting all edges with some color  $c \in \tilde{C}$  from  $G$  disconnects  $s$  and  $t$ . Motivated by applications in the design of robust networks, we introduce a family of problems called *colored cut games*. In these games, an attacker and a defender choose colors to delete and to protect, respectively, in an alternating fashion. It is the goal of the attacker to achieve a colored  $(s, t)$ -cut and the goal of the defender to prevent this. First, we show that for an unbounded number of alternations, colored cut games are PSPACE-complete. We then show that, even on subcubic graphs, colored cut games with a constant number  $i$  of alternations are complete for classes in the polynomial hierarchy whose level depends on  $i$ . To complete the dichotomy, we show that all colored cut games are polynomial-time solvable on graphs with degree at most two. Finally, we show that all colored cut games admit a polynomial kernel for the parameter  $k + \kappa_r$  where  $k$  denotes the total attacker budget and, for any constant  $r$ ,  $\kappa_r$  is the number of vertex deletions that are necessary to transform  $G$  into a graph where the longest path has length at most  $r$ . In the case of  $r = 1$ ,  $\kappa_1$  is the vertex cover number  $vc$  of the input graph and we obtain a kernel with  $\mathcal{O}(vc^2 k^2)$  edges. Moreover, we introduce an algorithm solving the most basic colored cut game, COLORED  $(s, t)$ -CUT, in  $2^{vc+k} n^{\mathcal{O}(1)}$  time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms; Theory of computation  $\rightarrow$  Graph algorithms analysis; Theory of computation  $\rightarrow$  Problems, reductions and completeness

**Keywords and phrases** Labeled Cut, Labeled Path, Network Robustness, Kernelization, PSPACE, Polynomial Hierarchy

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.30

**Funding** *Nils Morawietz*: Partially supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

*Frank Sommer*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project MAGZ, KO 3669/4-1.

**Acknowledgements** Some of the results of this work are also contained in the first author's Master thesis [21].



© Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer; licensed under Creative Commons License CC-BY

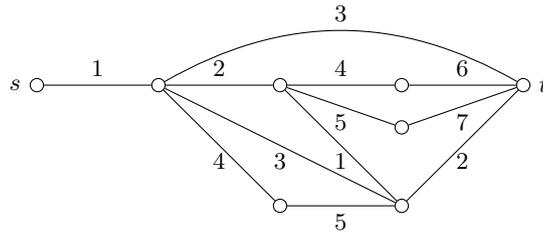
40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 30; pp. 30:1–30:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A colored cut game of two rounds on an edge-colored graph with seven colors: In round one, the defender may protect one color and the attacker may attack two colors. In round two, the defender can protect two colors, and the attacker can attack one color. For example, the defender may protect color 1, then the attacker may attack colors 2 and 3, then in round two, the defender may protect colors 4 and 5. The resulting graph has two  $(s, t)$ -paths containing the colors 1, 4, 5, 6 and 1, 4, 5, 7, respectively. Since the attacker may now only attack either 6 or 7, the defender wins.

## 1 Introduction

Many classic computational graph problems are motivated by applications in network robustness. A famous example is the problem of computing a minimum cut between two given vertices  $s$  and  $t$  in a simple undirected graph  $G = (V, E)$  [12, 17]. In some applications, a more realistic model for the robustness of a given network can be obtained by considering edge-colored graphs. Here, the input graph  $G$  comes with a coloring  $\ell : E \rightarrow C$  of the edges, where  $C$  is the set of colors. For example, in multilayer networks a failure of some link in a basic network layer may result in a failure of many seemingly unrelated links in a virtual network layer, because all of the virtual links rely on paths in the basic network that use the failed link [7]. This can be modeled by assigning edge colors. A failure of the resource represented by a color  $c$  then destroys all edges with color  $c$ . Thus, whether a failure scenario disconnects two given vertices depends directly on the colors of  $C$  that fail. More precisely, given  $s \in V$  and  $t \in V$ , a set  $\tilde{C} \subseteq C$  is a *colored  $(s, t)$ -cut* in  $G$  if every  $(s, t)$ -path contains at least one edge that has a color of  $\tilde{C}$ . For example, the color set  $\{2, 3, 4\}$  is a colored  $(s, t)$ -cut in Figure 1.

The size of the smallest colored  $(s, t)$ -cut then becomes an important network robustness parameter in scenarios modeled by colored graphs. Motivated by this fact, the problem of computing such a colored cut, called **COLORED  $(s, t)$ -CUT** in the following, has been studied intensively [4, 7, 8, 15, 22, 28, 31]. In contrast to the classic problem on uncolored graphs, **COLORED  $(s, t)$ -CUT** is NP-complete [7]. We may view **COLORED  $(s, t)$ -CUT** as formulated from the perspective of an attacker whose aim is to disconnect  $s$  and  $t$  using a minimum number of edge colors. A related  $(s, t)$ -connectivity problem is **LABELED PATH**, where we ask for a smallest color set  $\tilde{C} \subseteq C$  such that there is an  $(s, t)$ -path whose edges are only colored with colors from  $\tilde{C}$  [7, 15, 29]. **LABELED PATH** is NP-complete in general [29]; when every edge color occurs at most once it is simply **SHORTEST PATH** and thus solvable in polynomial time. In our scenario, **LABELED PATH** can be seen as motivated from the perspective of a defender who wants to secure a minimum number of edge colors in order to guarantee that  $s$  and  $t$  are connected.

We study colored cut games in which defender and attacker interact. This is motivated by typical studies in network security where an attacker (sometimes called red team) plays against a defender (sometimes called blue team) [20]. Such scenarios can be modelled using game-theoretic formalizations [14, 19, 25] as we do in this work. In the standard nomenclature [25], we study dynamic games with perfect information where the aim is to complete or to prevent a colored cut.

More precisely, we assume that there are two players that alternately choose colors. The colors chosen by the attacker are deleted from the graph while the colors chosen by the defender become safe which means that the attacker may not choose these colors in subsequent turns. In our model, for each turn the attacker and the defender have a fixed budget limiting the number of colors that they may choose. We study different versions of this game, Figure 1 shows an example. We distinguish, for example, whether the number of alternations between defender and attacker is constant or unbounded, whether the defender or the attacker starts, and whether we are interested in a winning strategy for the defender or the attacker. We refer to the family of these games as *colored cut games*.

COLORED  $(s, t)$ -CUT is the colored cut game where the attacker has one turn, the defender has none, and we ask if the attacker has a winning strategy. LABELED PATH can be seen as the colored cut game where the defender starts with a limited budget, followed by the attacker with unlimited budget, and we ask if the defender has a winning strategy. When the number of alternations between defender and attacker is unbounded, then we refer to the game as  $(DA)^*$  COLORED  $(s, t)$ -CUT ROBUSTNESS  $((DA)^*$ -CCR). The well-known SHANNON SWITCHING GAME [5, 6] which is polynomial-time solvable is the special case of  $(DA)^*$ -CCR where every edge color appears at most once and each player may choose one color in every turn.

**Our Results.** We study the complexity of colored cut games. In Section 3, we show that, in contrast to SHANNON SWITCHING GAME,  $(DA)^*$ -CCR is PSPACE-complete, and that for an increasing but constant number of alternations between the agents, the colored cut games are complete for complexity classes of increasing levels of the polynomial hierarchy.

In Section 4.1, we study how the structure of the input graph influences the complexity of the games. We show, for example, that all colored cut games are polynomial-time solvable on graphs with degree at most two and hard for different levels of the polynomial hierarchy on bipartite planar subcubic graphs. Finally, in Section 4.2 and Section 4.3 we study the parameterized complexity of colored cut games. Our main result is a polynomial-size problem kernel for all colored cut games parameterized by  $k + \kappa_r$ . Here  $k$  is the sum of all budgets of the attacker and  $\kappa_r$  is the number of vertex deletions that are needed to transform the input graph  $G$  into a graph where the longest path has length at most  $r$  (thus,  $\kappa_1$  is the vertex cover number  $vc$  of  $G$ ). More precisely, we show that for every constant  $r$  we can reduce any instance of a colored cut game in polynomial time to one with  $\mathcal{O}((\kappa_r)^2 k^{r+1})$  edges. This general kernelization result is somewhat surprising because for most parameters (including the vertex cover number,  $k$ , or  $|C|$ ) even the basic colored cut games COLORED  $(s, t)$ -CUT and LABELED PATH are unlikely to admit a polynomial kernelization [15, 18, 22, 31]; the first nontrivial kernelization for COLORED  $(s, t)$ -CUT (with respect to a rather large parameter) was provided, to the best of our knowledge, in our companion work on COLORED  $(s, t)$ -CUT [22]. We are not aware of other studies of kernelization for PSPACE-hard problems. In addition to the kernelization, we develop an algorithm solving COLORED  $(s, t)$ -CUT in  $2^{vc+k} n^{\mathcal{O}(1)}$  time. One of the main tools in our hardness proofs and algorithms is the notion of colored-cut-equivalence. This notion may be of general interest for the study of colored cuts in graphs. We define colored-cut-equivalence in Section 2, where we give the formal definition of the colored cut games. Due to lack of space, several proofs are deferred to a long version of this article.

## 2 Basic Definitions and Colored-Cut-Equivalence

**Notation.** For integers  $j$  and  $k, j \leq k$ , we denote with  $[j, k]$  the set  $\{r \mid j \leq r \leq k\}$ . For a set  $S$  and an integer  $k$ , we let  $\binom{S}{k}$  denote the family of all size- $k$  subsets of  $S$ . A (simple undirected) graph  $G = (V, E)$  consists of a finite set of vertices  $V(G) := V$  and a set of edges  $E(G) := E \subseteq \binom{V}{2}$  and we denote  $n := |V|$  and  $m := |E|$ . For  $V' \subseteq V$ , we denote with  $G[V'] := (V', E \cap \binom{V'}{2})$  the subgraph of  $G$  induced by  $V'$  and with  $G - V' := G[V \setminus V']$  the graph obtained from  $G$  by deleting  $V'$ . Analogously, we let  $G - E' := (V, E \setminus E')$  denote the graph obtained by deleting the edge set  $E' \subseteq E$ . We denote with  $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$  the neighborhood of a vertex  $v$  in  $G$  and we denote with  $\deg_G(v) := |N_G(v)|$  the degree of  $v$  in  $G$ . If  $G$  is clear from the context, we may omit the subscript.

A sequence of vertices  $P = (v_1, \dots, v_k)$  is a path or  $(v_1, v_k)$ -path of length  $k$  in  $G$  if  $\{v_i, v_{i+1}\} \in E(G)$  for all  $1 \leq i < k$ . If  $v_i \neq v_j$  for all  $i \neq j$ , then we call  $P$  vertex-simple. If not mentioned otherwise, we only consider vertex-simple paths. We denote with  $V(P)$  the vertices of  $P$  and with  $E(P)$  the edges of  $P$ . A subset  $V' \subseteq V$  is called a connected component of  $G$  if  $V'$  is a maximal set of vertices such that there is at least one  $(u, v)$ -path in  $G$  for pairwise distinct  $u, v \in V'$ .

**Parameterized Complexity.** For the definition of classical complexity classes such as PSPACE or  $\Sigma_2^P$ , we refer to the literature [2]. Parameterized complexity theory aims at a fine-grained analysis of the computational complexity of hard problems [9, 11, 16, 24]. A parameterized problem  $L$  is a subset of  $\Sigma^* \times \mathbb{N}$ , where the first component is the input and the second is the parameter. A parameterized problem is fixed-parameter tractable (FPT) if every instance  $(I, k)$  can be solved in  $f(k) \cdot |I|^{O(1)}$  time where  $f$  is a computable function depending only on  $k$ ; an algorithm with this running time is called FPT algorithm. A parameterized problem is in XP if every instance can be solved in  $|I|^{g(k)}$  time for some computable function  $g$ . The complexity classes W[1] and W[2] are basic classes of presumed parameterized intractability, that is, it is assumed that problems that are hard for W[1] or W[2] have no fixed-parameter algorithm. Hardness for W[1] or W[2] is shown via parameterized reductions. A parameterized reduction of a parameterized problem  $L$  to a parameterized problem  $L'$  is an algorithm that for each instance  $(I, k)$  of  $L$  computes in  $f(k) \cdot |I|^{O(1)}$  time an equivalent instance  $(I', k')$  of  $L'$  such that  $k' \leq g(k)$  for some computable function  $g$ . A parameterized reduction is a polynomial parameter transformation if  $g(k)$  is a polynomial function.

A main tool to achieve fixed-parameter algorithms is reduction to a problem kernel or problem kernelization. A problem kernelization for a parameterized problem  $L$  is a polynomial-time algorithm that computes for every instance  $(I, k)$  an equivalent instance  $(I', k')$  such that  $|I'| \leq g(k)$  and  $k' \leq f(k)$  for computable functions  $f$  and  $g$ . If  $g$  and  $f$  are polynomials then, we call it a polynomial problem kernelization.

**Colored Cut Games.** An edge-colored graph with terminals (or colored graph) is a 5-tuple  $\mathcal{H} = (G = (V, E), s, t, C, \ell)$  where  $G$  is an undirected graph,  $s \in V$  and  $t \in V$  are the terminals,  $C$  is a set of colors and  $\ell : E \rightarrow C$  is an edge coloring. We denote with  $|\mathcal{H}| := |G| + |C| + |\ell| = |V| + 2|E| + |C|$  the size of a colored graph.

For a graph  $G = (V, E)$  and two vertices  $s \in V$  and  $t \in V$ , we call an edge set  $E' \subseteq E$  an  $(s, t)$ -(edge-)cut in  $G$  if  $s$  and  $t$  are in different connected components in  $G - E'$ . Let  $\mathcal{H} = (G, s, t, C, \ell)$  be a colored graph. For a path  $P$  in  $G$ , we let  $\ell(P) := \ell(E(P))$  denote the set of colors of the edges on this path. We say that  $\tilde{C} \subseteq C$  is a colored  $(s, t)$ -cut in  $G$  if  $\ell(P) \cap \tilde{C} \neq \emptyset$  for every  $(s, t)$ -path  $P$  in  $G$ . We say that  $\tilde{C} \subseteq C$  is a colored  $(s, t)$ -connector in  $G$  if there is an  $(s, t)$ -path  $P$  in  $G$  with  $\ell(P) \subseteq \tilde{C}$ .

We now formally define all colored cut games. Since the outcome of the game is decided after the last turn of the attacker, all colored cut games end with a turn of the attacker. In the most general problem variant, stated below, we allow an unbounded number of alternations between the defender D and the attacker A.

(DA)\* COLORED  $(s, t)$ -CUT ROBUSTNESS ((DA)\*-CCR)

**Input:** A colored graph  $(G = (V, E), s, t, C, \ell)$ , and two vectors  $\vec{d} := (d_1, \dots, d_i) \in \mathbb{N}^i$  and  $\vec{a} := (a_1, \dots, a_i) \in \mathbb{N}^i$  such that  $\sum_{j=1}^i (d_j + a_j) \leq |C|$ .

**Question:** Is it true that  $\exists D_1 \in \binom{C}{d_1}. \forall A_1 \in \binom{C \setminus D_1}{a_1}. \exists D_2 \in \binom{C \setminus (D_1 \cup A_1)}{d_2}. \dots \forall A_i \in \binom{C \setminus (\bigcup_{j=1}^{i-1} (D_j \cup A_j) \cup D_i)}{a_i}$  : the set  $\bigcup_{j=1}^i A_j$  is not a colored  $(s, t)$ -cut in  $G$ ?

In (DA)\*-CCR we ask if the defender has a *winning strategy*. When the number of turns  $i \geq 1$  is a constant and not part of the input, we define the problems (DA)<sup>i</sup> COLORED  $(s, t)$ -CUT ROBUSTNESS ((DA)<sup>i</sup>-CCR).

If the attacker starts the game, that is, if  $d_1 = 0$ , we define the problems A(DA)<sup>i</sup> COLORED  $(s, t)$ -CUT ROBUSTNESS (A(DA)<sup>i</sup>-CCR) for all constant  $i \geq 0$ . For all these problems we also define the complement problems in which we ask if there is a winning strategy for the attacker.

(DA)\* COLORED  $(s, t)$ -CUT VULNERABILITY ((DA)\*-CCV)

**Input:** A colored graph  $(G = (V, E), s, t, C, \ell)$ , and two vectors  $\vec{d} := (d_1, \dots, d_i) \in \mathbb{N}^i$  and  $\vec{a} := (a_1, \dots, a_i) \in \mathbb{N}^i$  such that  $\sum_{j=1}^i (d_j + a_j) \leq |C|$ .

**Question:** Is it true that  $\forall D_1 \in \binom{C}{d_1}. \exists A_1 \in \binom{C \setminus D_1}{a_1}. \forall D_2 \in \binom{C \setminus (D_1 \cup A_1)}{d_2}. \dots \exists A_i \in \binom{C \setminus (\bigcup_{j=1}^{i-1} (D_j \cup A_j) \cup D_i)}{a_i}$  : the set  $\bigcup_{j=1}^i A_j$  is a colored  $(s, t)$ -cut in  $G$ ?

Analogously, if the number of alternations is a constant, then we define the variants (DA)<sup>i</sup>-CCV and A(DA)<sup>i</sup>-CCV. We refer to all problems defined above as *colored cut games*.

COLORED  $(s, t)$ -CUT is equivalent to A(DA)<sup>0</sup>-CCV and LABELED PATH is the special case of (DA)<sup>1</sup>-CCR where  $a_1 = |C| - d_1$ . Moreover, for all  $i \geq 1$ , A(DA)<sup>i-1</sup>-CCR is the special case of (DA)<sup>i</sup>-CCR where the budget of the first defender turn is zero and (DA)<sup>i</sup>-CCR is the special case of A(DA)<sup>i</sup>-CCR where the budget of the first attacker turn is zero. Hence, COLORED  $(s, t)$ -CUT is a special case of all the problems (DA)<sup>i</sup>-CCV and A(DA)<sup>i</sup>-CCV.

**Colored-Cut-Equivalence.** We let  $\mathcal{C}(\mathcal{H}) := \{\ell(P) \mid P \text{ is an } (s, t)\text{-path in } G\}$  denote the family of color sets of  $(s, t)$ -paths in  $G$ .

► **Observation 2.1.** *The set of colors  $\tilde{C} \subseteq C$  is a colored  $(s, t)$ -cut in  $G$  if and only if  $\tilde{C}$  is a hitting set for  $\mathcal{C}(\mathcal{H})$ , that is, if  $\tilde{C} \cap C' \neq \emptyset$  for all  $C' \in \mathcal{C}(\mathcal{H})$ .*

Moreover,  $\tilde{C}$  is a colored  $(s, t)$ -connector in  $G$  if and only if there is  $C' \in \mathcal{C}(\mathcal{H})$  such that  $C' \subseteq \tilde{C}$ .

To argue concisely that two instances of some colored cut game are equivalent, we introduce the following definition.

► **Definition 2.1.** *Two colored graphs  $\mathcal{H} = (G, s, t, C, \ell)$  and  $\mathcal{H}' = (G', s', t', C, \ell')$  are colored-cut-equivalent if for every  $L_1 \in \mathcal{C}(\mathcal{H}) \cup \mathcal{C}(\mathcal{H}')$  there exists some  $L_2 \in \mathcal{C}(\mathcal{H}) \cap \mathcal{C}(\mathcal{H}')$  such that  $L_2 \subseteq L_1$ .*

Observe that  $\mathcal{H}$  and  $\mathcal{H}'$  are colored-cut-equivalent if for every  $(s, t)$ -path  $P$  in  $G$  there is an  $(s', t')$ -path  $P'$  in  $G'$  such that  $\ell'(P') \subseteq \ell(P)$  and vice versa. Thus, intuitively, only the color sets in  $\mathcal{C}(\mathcal{H}) \cap \mathcal{C}(\mathcal{H}')$  are relevant for colored  $(s, t)$ -cuts. The following lemma shows that Definition 2.1 gives us the intended property.

► **Lemma 2.2.** *Let  $\mathcal{H} = (G, s, t, C, \ell)$  and  $\mathcal{H}' = (G', s', t', C, \ell')$  be two colored-cut-equivalent graphs, then  $\tilde{C} \subseteq C$  is a colored  $(s, t)$ -cut in  $G$  if and only if  $\tilde{C}$  is a colored  $(s', t')$ -cut in  $G'$ .*

**Proof.** Due to symmetry, we only show one direction. Let  $\tilde{C}$  be a colored  $(s, t)$ -cut in  $G$ , then  $\tilde{C} \cap L_2 \neq \emptyset$  for all  $L_2 \in \mathcal{C}(\mathcal{H}) \cap \mathcal{C}(\mathcal{H}')$ . We show  $\tilde{C} \cap L_1 \neq \emptyset$  for all  $L_1 \in \mathcal{C}(\mathcal{H}')$ . Let  $L_1 \in \mathcal{C}(\mathcal{H}')$ , then there is some  $L_2 \in \mathcal{C}(\mathcal{H}) \cap \mathcal{C}(\mathcal{H}')$  with  $L_2 \subseteq L_1$  since  $\mathcal{H}$  and  $\mathcal{H}'$  are colored-cut-equivalent. Hence,  $L_1 \cap \tilde{C} \supseteq L_2 \cap \tilde{C} \neq \emptyset$  and therefore  $\tilde{C}$  is a colored  $(s', t')$ -cut in  $G'$ . ◀

► **Corollary 2.3.** *Two instances  $I = (\mathcal{H}, \vec{d}, \vec{a})$  and  $I' = (\mathcal{H}', \vec{d}, \vec{a})$  of any colored cut game are equivalent if  $\mathcal{H}$  and  $\mathcal{H}'$  are colored-cut-equivalent.*

The following lemmas will be useful for proving hardness on restricted input graphs.

► **Lemma 2.4.** *For every colored graph  $\mathcal{H} = (G, s, t, C, \ell)$ , one can compute in polynomial time a colored-cut-equivalent graph  $\mathcal{H}' = (G', s', t', C, \ell')$  such that  $G'$  is bipartite.*

► **Lemma 2.5.** *Let  $\mathcal{H} = (G, s, t, C, \ell)$  be a colored graph and let  $\alpha \in C$  be a color that occurs on every  $(s, t)$ -path in  $\mathcal{H}$ . Then, one can compute in polynomial time a colored-cut-equivalent graph  $\mathcal{H}' = (G', s', t', C, \ell')$  such that  $G'$  has a maximum degree of three.*

### 3 Classic Complexity of Colored Cut Games

#### 3.1 Unbounded Number of Alternations

We first show that colored cut games are PSPACE-complete if the number of alternations between attacker and defender is unbounded by reducing from the PSPACE-complete COMPETITIVE HITTING SET [26].

► **Theorem 3.1.**  *$(DA)^*$ -CCR and  $(DA)^*$ -CCV are PSPACE-complete on planar graphs even if each budget is one.*

**Proof.**  $(DA)^*$ -CCR and  $(DA)^*$ -CCV can obviously be solved within polynomial space by a standard search tree algorithm that alternately chooses the colors for the defender and the attacker. Thus, it remains to show PSPACE-hardness. To this end we give a polynomial-time reduction from a competitive version of HITTING SET which is PSPACE-complete [26].

COMPETITIVE HITTING SET (CHS)

**Input:** A universe  $\mathcal{U}$  with  $|\mathcal{U}| = 2i$  and a collection  $\mathcal{F}$  of non-empty subsets of  $\mathcal{U}$ .

**Question:** Is it true that  $\forall d_1 \in \mathcal{U}. \exists a_1 \in \mathcal{U} \setminus \{d_1\}. \forall d_2 \in \mathcal{U} \setminus \{d_1, a_1\}. \dots \exists a_i \in \mathcal{U} \setminus \left( \bigcup_{j=1}^{i-1} \{d_j, a_j\} \cup \{d_i\} \right) : F \cap \{a_j \mid 1 \leq j \leq i\} \neq \emptyset$  for all  $F \in \mathcal{F}$ ?

This problem can be seen as a game between two agents where every agent selects an unselected element of the universe in each turn. The game ends when there is no unselected element of the universe remaining and the second player wins if he intersects every subset  $F \in \mathcal{F}$  with the elements he chose. Otherwise, the first player wins. We ask if the second player has a winning strategy.

Given an instance  $I = (\mathcal{U}, \mathcal{F})$  of COMPETITIVE HITTING SET, we describe how to construct an equivalent instance  $I' = (G = (V, E), s, t, C, \ell)$  of  $(\text{DA})^*$ -CCV in polynomial time. We set  $C := \mathcal{U}$  and start with an empty graph only containing distinct vertices  $s$  and  $t$ . For every  $F \in \mathcal{F}$  we add an  $(s, t)$ -path  $P_F$  such that  $\ell(P_F) = F$  and where all vertices of  $P_F$  except  $s$  and  $t$  are new. Thus, for every  $(s, t)$ -path  $P$  in  $G$  there is  $F \in \mathcal{F}$  such that  $\ell(P) = F$ . Consequently,  $A \subseteq \mathcal{U}$  intersects every  $F \in \mathcal{F}$  if and only if  $A$  is a colored  $(s, t)$ -cut in  $G$ .

Hence, a winning strategy for the attacker in the  $(\text{DA})^*$ -CCV instance  $I'$  is also a winning strategy for the second player in the COMPETITIVE HITTING SET instance  $I$  and vice versa. Therefore,  $I$  is a yes-instance of COMPETITIVE HITTING SET if and only if  $I'$  is a yes-instance of  $(\text{DA})^*$ -CCV. Since the class of PSPACE-complete problems is closed under complement,  $(\text{DA})^*$ -CCR where the budget in every turn is one is also PSPACE-complete. ◀

### 3.2 Bounded Number of Alternations

Next, we analyze the complexity of  $(\text{DA})^i$ -CCR and  $\text{A}(\text{DA})^i$ -CCR. To this end, recall that  $(\text{DA})^i$ -CCR asks if the defender has a winning strategy when the defender starts and both agents have exactly  $i$  turns for some constant  $i$ .

► **Lemma 3.2.** *For all  $i \geq 1$ ,  $(\text{DA})^i$ -CCV is  $\Pi_{2i}^P$ -hard and  $(\text{DA})^i$ -CCR is  $\Sigma_{2i}^P$ -hard even on planar graphs.*

To prove Lemma 3.2, we reduce  $\text{QSAT}_{2i}$  which we will state using the following notation. For a set of boolean variables  $Z$ , we define the set of *literals*  $\mathcal{L}(Z) := Z \cup \{\neg z \mid z \in Z\}$ . A subset of literals  $\tilde{Z} \subseteq \mathcal{L}(Z)$  is an *assignment* of  $Z$  if  $|\{z, \neg z\} \cap \tilde{Z}| = 1$  for all  $z \in Z$ . For a subset  $X \subseteq Z$  of variables, we denote with  $\tau_Z(X) := X \cup \{\neg z \mid z \in Z \setminus X\}$  the assignment of  $Z$  where all variables of  $X$  occur positively and all variables of  $Z \setminus X$  occur negatively. Given an assignment  $\tilde{Z}$  and a *clause*  $\phi \in \binom{\mathcal{L}(Z)}{3}$  we say that  $\tilde{Z}$  *satisfies*  $\phi$  (denoted by  $\tilde{Z} \models \phi$ ) if  $\phi \cap \tilde{Z} \neq \emptyset$ . Analogously,  $\tilde{Z}$  satisfies a set  $\Phi \subseteq \binom{\mathcal{L}(Z)}{3}$  of clauses (denoted by  $\tilde{Z} \models \Phi$ ) if  $\tilde{Z} \models \phi$  for all  $\phi \in \Phi$ .

**Proof sketch.** We reduce  $\text{QSAT}_{2i}$ , which is  $\Pi_{2i}^P$ -hard [2], to  $(\text{DA})^i$ -CCV.

$\text{QSAT}_{2i}$

**Input:** A set  $\Phi$  of clauses in 3-CNF over the set of variables  $Z$  and a partition  $(X_1, Y_1, \dots, X_i, Y_i)$  of  $Z$ .

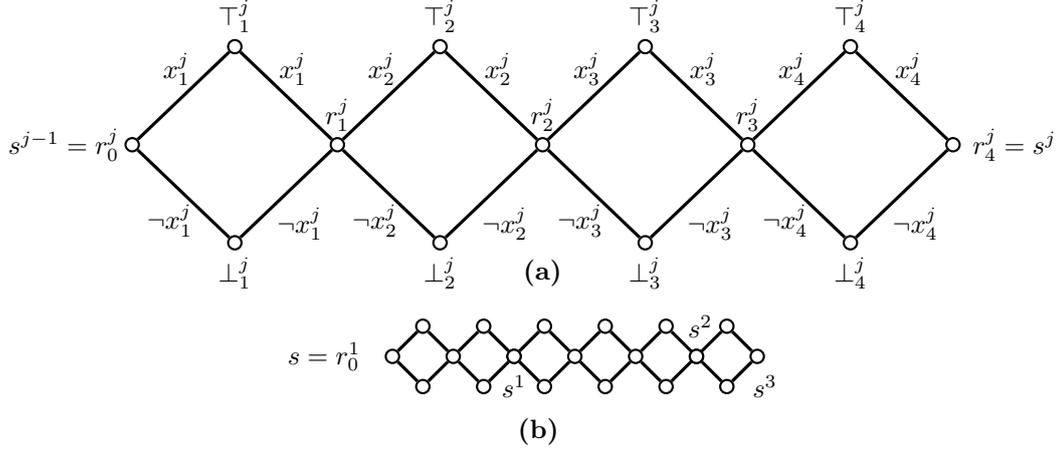
**Question:** Is it true that  $\forall \tilde{X}_1 \subseteq X_1. \exists \tilde{Y}_1 \subseteq Y_1. \dots \forall \tilde{X}_i \subseteq X_i. \exists \tilde{Y}_i \subseteq Y_i : \tau_Z(\tilde{X}_1 \cup \tilde{Y}_1 \cup \dots \cup \tilde{X}_i \cup \tilde{Y}_i) \models \Phi$ ?

$\text{QSAT}_{2i}$  can be seen as a two-player game where Player 1 and Player 2 choose an assignment for  $X_j$  and  $Y_j$ , respectively, in their  $j$ th turn. We ask if Player 2 has a winning strategy, that is, if the combined assignment satisfies  $\Phi$ .

Given an instance  $I' = (Z, \Phi, X_1, Y_1, \dots, X_i, Y_i)$  of  $\text{QSAT}_{2i}$ , we construct an instance  $I = (\mathcal{H}, \vec{d}, \vec{a})$  of  $(\text{DA})^i$ -CCV as follows. Let  $X_j = \{x_k^j \mid 1 \leq k \leq |X_j|\}$ ,  $Y_j = \{y_k^j \mid 1 \leq k \leq |Y_j|\}$  for all  $1 \leq j \leq i$  and let  $\mathcal{L} := \mathcal{L}(Z)$ . We can assume without loss of generality that  $|X_j| \geq 2$  for all  $j \in [2, i]$  and  $|Y_j| \geq 2$  for all  $j \in [1, i]$ .

We set  $C := \mathcal{L}$  and force the defender and the attacker to choose an assignment of the variables of  $X_j$  and  $X_j \cup Y_j$ , respectively, in their  $j$ th turn, otherwise they will lose.

The graph consists of three parts: the variable gadgets for the defender, the variable gadgets for the attacker and a gadget for the evaluation of the clauses. To this end, we define  $G := (V, E)$  with  $V := V_d \cup V_a \cup V_\Phi$  and  $E := E_d \cup E_a \cup E_\Phi$  where  $V_d, E_d$  and  $V_a, E_a$  are the variable gadgets for the defender and attacker, respectively, and  $V_\Phi, E_\Phi$  is the gadget



■ **Figure 2** (a) The gadget for the defender for the variables of  $X_j$  with  $|X_j| = 4$ . (b) The graph  $G_D = (V_D, E_D)$  where  $|X_1| = 2$ ,  $|X_2| = 3$ , and  $|X_3| = 1$ .

for the evaluation of the clauses. First, we introduce the variable gadget for the defender, shown in Figure 2:

- $V_d := \{r_0^j \mid 1 \leq j \leq i\} \cup \{r_k^j, \top_k^j, \perp_k^j \mid 1 \leq j \leq i, 1 \leq k \leq |X_j|\}$
- $E_d := \left\{ \{r_{k-1}^j, \top_k^j\}, \{r_{k-1}^j, \perp_k^j\}, \{\top_k^j, r_k^j\}, \{\perp_k^j, r_k^j\} \mid 1 \leq j \leq i, 1 \leq k \leq |X_j|\right\}$ ,
- $\ell(\{r_{k-1}^j, \top_k^j\}) := \ell(\{\top_k^j, r_k^j\}) := x_k^j$ ,
- $\ell(\{r_{k-1}^j, \perp_k^j\}) := \ell(\{\perp_k^j, r_k^j\}) := \neg x_k^j$ ,

where  $r_{|X_j|}^j = r_0^{j+1}$  for all  $1 \leq j < i$ . In the following, let  $s := s^0 := r_0^1$  and  $s^j := r_{|X_j|}^j$  for all  $j \in [1, i]$ . The vertex  $s_j$  is a common vertex of the gadgets for the attacker and defender. The idea is that in his  $j$ th turn the defender has to choose an assignment of the variables of  $X_j$ , or otherwise the attacker wins by taking at most two colors in his next turn. Next, we define the gadgets for the attacker:

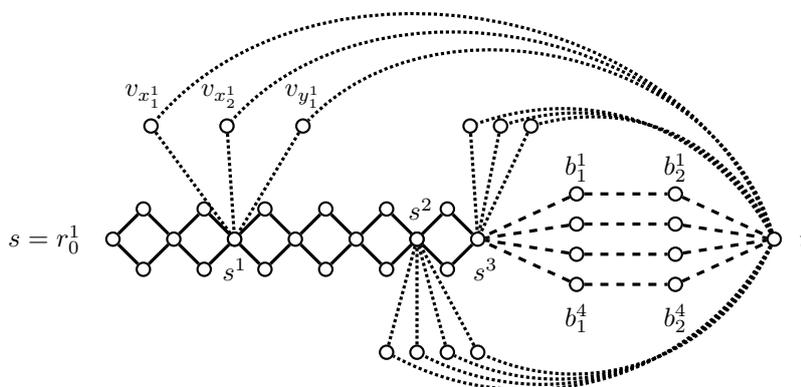
- $V_a := \{t\} \cup \{v_x \mid x \in Z\}$ ,
- $E_a := \left\{ \{s^j, v_x\}, \{v_x, t\} \mid 1 \leq j \leq i, x \in X_j \cup Y_j \right\}$ ,
- $\ell(\{s^j, v_x\}) := x$ , and  $\ell(\{v_x, t\}) := \neg x$  for all  $j \in [1, i], x \in X_j \cup Y_j$ .

The idea is that either the color set chosen by the attacker in his  $j$ th turn is an assignment of the variables of  $X_j \cup Y_j$ , or the defender wins by choosing two colors in his next turn. Since a player can only choose colors that were not chosen before, the assignment for the variables of  $X_j$  of the attacker is the complement of the assignment on the variables of  $X_j$  of the defender.

Finally, we define the clause gadget. To model each clause  $\phi \in \Phi$ , we add an  $(s^i, t)$ -path  $P$  with  $\ell(P) = \phi$ . Formally, the gadget is defined as follows. We fix an ordering on every clause  $\phi_j \in \Phi$  and denote with  $\phi_j(y)$  the  $y$ th literal of  $\phi_j$  and add

- $V_\Phi := \{b_1^j, b_2^j \mid 1 \leq j \leq |\Phi|\}$ ,
- $E_\Phi := \left\{ \{s^i, b_1^j\}, \{b_1^j, b_2^j\}, \{b_2^j, t\} \mid 1 \leq j \leq |\Phi|\right\}$ ,
- $\ell(\{s^i, b_1^j\}) := \phi_j(1)$ ,
- $\ell(\{b_1^j, b_2^j\}) := \phi_j(2)$ , and
- $\ell(\{b_2^j, t\}) := \phi_j(3)$ .

The final graph can be seen in Figure 3. We set  $d_j := |X_j|$  and  $a_j := |X_j| + |Y_j|$  for all  $j \in [1, i]$ . This completes the construction.



■ **Figure 3** The construction for an instance with  $|\Phi| = 4$ ,  $|X_1| = |Y_3| = 2$ ,  $|Y_1| = |Y_2| = |X_3| = 1$ , and  $|X_2| = 3$ . Solid edges belong to  $E_d$ , dotted edges belong to  $E_a$ , dashed edges belong to  $E_\Phi$ . The clause gadget is connected with  $s^3$  and  $t$ .

Before we show the equivalence between  $I$  and  $I'$ , we make some observations about winning strategies. The following establishes the link between sensible choices of color sets and partial assignments for variables in  $Z$ : Let  $j \in [1, i]$  and let  $D_j \subseteq C$  be the set of colors the defender chooses in his  $j$ th turn. We call  $D_j$  *nice* if  $D_j$  is an assignment for  $X_j$ . Analogously, let  $A_j \subseteq C$  be the set of colors the attacker chooses in his  $j$ th turn. We call  $A_j$  *nice* if  $A_j$  is an assignment for  $X_j \cup Y_j$ .

▷ **Claim 3.3.** For both players, it is never part of a winning strategy to be the first to choose a set of colors which is not nice.

Hence, we can assume that both players will only choose nice sets of colors.

▷ **Claim 3.4.** Let  $D_j, A_j$  be nice for all  $j \in [1, i]$  and  $\tilde{A} := \bigcup_{j=1}^i A_j$ , then  $\tilde{A}$  is a colored  $(s, t)$ -cut in  $G$  if and only if  $\tilde{A} \models \Phi$ .

Using these claims, we show that the  $\text{QSAT}_{2i}$  instance is a yes-instance if and only if the constructed  $(\text{DA})^i$ -CCV instance is a yes-instance.

( $\Rightarrow$ ) Assume that  $\forall \tilde{X}_1 \subseteq X_1. \exists \tilde{Y}_1 \subseteq Y_1. \dots \forall \tilde{X}_i \subseteq X_i. \exists \tilde{Y}_i \subseteq Y_i. \tau_Z(\tilde{X}_1 \cup \tilde{Y}_1 \cup \dots \cup \tilde{X}_i \cup \tilde{Y}_i) \models \Phi$  is true. Then, there are functions  $f_k : \mathbb{P}(\bigcup_{j=1}^k \tilde{X}_j) \rightarrow \mathbb{P}(Y_k)$  for all  $k \in [1, i]$  such that  $\forall \tilde{X}_1 \subseteq X_1. \dots \forall \tilde{X}_i \subseteq X_i. \tau_Z(\tilde{X}_1 \cup f_1(\tilde{X}_1) \cup \dots \cup \tilde{X}_i \cup f_i(\bigcup_{k=1}^i \tilde{X}_k)) \models \Phi$  is true [3]. Herein,  $\mathbb{P}$  denotes the powerset. The functions  $f_1, \dots, f_i$  are called *Skolem functions* and can be seen as the winning strategy of Player 2 in the  $\text{QSAT}_{2i}$  instance. We will use these functions to describe a winning strategy for the attacker in the  $(\text{DA})^i$ -CCV instance iteratively. Let  $D_1$  be the color set chosen by the defender in his first turn. If  $D_1$  is not nice then, due to Claim 3.3, the attacker has a winning strategy. So, we assume that  $D_1$  is nice. Then,  $D_1$  is an assignment for  $X_1$ . Let  $\overline{D}_1 := X_1 \setminus D_1$ , that is, the complement assignment of  $D_1 \cap X_1$ . We set  $A_1 := \tau_{X_1 \cup Y_1}(\overline{D}_1 \cup f_1(\overline{D}_1))$  which is nice and disjoint from  $D_1$ .

After this initial choice, the winning strategy for the attacker works as follows. Let  $j \in [2, i]$  such that  $D_r$  and  $A_r$  are nice for all  $r \in [1, j-1]$ . Let  $D_j$  be the color set chosen by the defender in his  $j$ th turn. If  $D_j$  is not nice then, due to Claim 3.3, the attacker has a winning strategy. So, we assume that  $D_j$  is nice. Then,  $D_j$  is an assignment for  $X_j$ . Let  $\overline{D}_r := X_r \setminus D_r$ , that is,

the complement assignment of  $D_r$  for all  $r \in [1, j]$ . We set  $A_j := \tau_{X_j \cup Y_j}(\overline{D}_j \cup f_j(\bigcup_{r=1}^j \overline{D}_r))$ . Observe that  $A_j$  is also nice. Hence, we can assume that  $D_j$  is nice and  $A_j$  is nice and defined as described for all  $j \in [1, i]$ .

It remains to show that  $\tilde{A}_i := \bigcup_{j=1}^i A_j$  is a colored  $(s, t)$ -cut in  $G$ . Since we assumed that  $\forall \tilde{X}_1 \subseteq X_1 \cdots \forall \tilde{X}_i \subseteq X_i. \tau_Z(\tilde{X}_1 \cup f_1(\tilde{X}_1) \cup \cdots \cup \tilde{X}_i \cup f_i(\bigcup_{k=1}^i \tilde{X}_k)) \models \Phi$  is true, it follows that  $\tilde{A}_i = \tau_Z(\overline{D}_1 \cup f_1(\overline{D}_1) \cup \cdots \cup \overline{D}_i \cup f_i(\bigcup_{k=1}^i \overline{D}_k)) \models \Phi$ . Therefore,  $\tilde{A}_i$  is a colored  $(s, t)$ -cut in  $G$  due to Claim 3.4. Hence, the attacker has a winning strategy.

( $\Leftarrow$ ) The proof of this direction is deferred to the the long version of this article.

Hence,  $I$  is a yes-instance of  $(DA)^i$ -CCV if and only if  $I'$  is a yes-instance of  $QSAT_{2i}$ . Therefore,  $(DA)^i$ -CCV is  $\Pi_{2i}^P$ -hard. Since  $(DA)^i$ -CCR is the complement problem of  $(DA)^i$ -CCV, it follows that  $(DA)^i$ -CCR is  $\Sigma_{2i}^P$ -hard.  $\blacktriangleleft$

Lemma 3.2 is the main step to prove the following.

► **Theorem 3.5.** *For all  $i \geq 0$ ,  $A(DA)^i$ -CCR is  $\Pi_{2i+1}^P$ -complete and for all  $i \geq 1$ ,  $(DA)^i$ -CCR is  $\Sigma_{2i}^P$ -complete even on planar graphs.*

## 4 Restricted Instances and Parameterizations

We now take a closer look at the classic complexity of  $(DA)^i$ ,  $A(DA)^i$ , and  $(DA)^*$ -CCR on restricted instances. First, we obtain a complexity dichotomy with regard to the maximum degree and strengthen our hardness results from Section 3.1 to restricted graph classes. Second, we analyze a restricted class of colored graphs for which COLORED  $(s, t)$ -CUT is polynomial-time-solvable and show that  $DA$ -CCR is NP-complete on these restricted colored graphs. Finally, we investigate the parameterized complexity and describe how to obtain polynomial kernel for all colored cut games by combining the budget with structural graph parameters.

### 4.1 Restricted Instances

First, we show that the classic complexity of all colored cut games is the same even on bipartite planar graphs. Second, we show that  $(DA)^i$ -CCR,  $A(DA)^i$ -CCR,  $i \geq 1$ , and  $(DA)^*$ -CCR can be solved in polynomial time on graphs with maximum degree at most two but cannot be solved in polynomial time on graphs with maximum degree at least three, unless  $P = NP$ .

By Theorem 3.5,  $(DA)^i$ -CCV and  $A(DA)^i$ -CCV are hard even on planar graphs. Given a planar graph, we can replace it with a bipartite planar colored-cut-equivalent graph in polynomial time due to Lemma 2.4. By Corollary 2.3, this gives an equivalent instance.

► **Corollary 4.1.** *For all  $i \geq 1$ ,  $(DA)^i$ -CCV is  $\Pi_{2i}^P$ -complete and for all  $i \geq 0$ ,  $A(DA)^i$ -CCV is  $\Sigma_{2i+1}^P$ -complete even on bipartite planar graphs.*

► **Theorem 4.2.** *Let  $i \geq 1$ . The problems  $(DA)^i$ -CCR,  $A(DA)^i$ -CCR, and  $(DA)^*$ -CCR can be solved in polynomial time on graphs with a maximum degree of at most two. On bipartite planar graphs with a maximum degree of at least three,  $(DA)^i$ -CCR and  $A(DA)^i$ -CCR are  $\Sigma_{2i}^P$ -hard and  $(DA)^*$ -CCR is PSPACE-hard.*

Second, we analyze the complexity of  $(DA)^1$ -CCR on instances where every color appears in at most two  $(s, t)$ -paths. In this case, COLORED  $(s, t)$ -CUT can be solved in polynomial time [7, 17, 27]. In contrast, we will show that  $(DA)^1$ -CCR is NP-complete. Hence, for any  $i \geq 1$ ,  $(DA)^i$ -CCR and  $A(DA)^i$ -CCR cannot be solved in polynomial time on these restricted colored graphs, unless  $P = NP$ . We show NP-completeness via reduction from MATCHING INTERDICTION which is NP-hard [30].

► **Theorem 4.3.**  $(DA)^1$ -CCR is NP-complete and W[1]-hard when parameterized by  $d_1$  even if every color appears in at most two  $(s, t)$ -paths.

## 4.2 Parameterization by the Full Budget and the Number of Colors

In this section we analyze the parameterized complexity of the colored cut games. Next, we investigate budget-related parameters. For an instance  $I = (\mathcal{H}, \vec{d}, \vec{a})$  of a colored cut game we denote with  $b(I) := \sum_{x=1}^i (d_x + a_x)$  the sum of all budgets and with  $k := \sum_{x=1}^i a_x$  the total budget of the attacker. First, we investigate the parameter  $b(I)$ . COLORED  $(s, t)$ -CUT is W[2]-hard when parameterized by  $k = b(I)$  [7]. We extend this hardness result to all colored cut games. Moreover, we show that all colored cut games are fixed-parameter tractable and do not admit polynomial kernels when parameterized  $|C|$ .

► **Proposition 4.4.**  $(DA)^i$ -CCR,  $i \geq 1$ ,  $A(DA)^i$ -CCR,  $i \geq 0$ , and  $(DA)^*$ -CCR parameterized by  $b(I)$  are coW[2]-hard and can be solved in  $\mathcal{O}(|C|^{b(I)}(n+m))$  time.

By definition,  $b(I) \leq |C|$ . Hence, the described algorithm of Proposition 4.4 with a running time of  $\mathcal{O}(|C|^{b(I)}(n+m))$  also implies an FPT-algorithm when parameterized by  $|C|$ .

► **Corollary 4.5.**  $(DA)^i$ -CCR,  $A(DA)^{i-1}$ -CCR,  $i \geq 1$ , and  $(DA)^*$ -CCR can be solved in time  $\mathcal{O}(\min(|C|^{|C|}, 2^{2^{|C|}})(n+m))$  and do not admit a polynomial kernel when parameterized by  $|C|$ , unless  $\text{NP} \subseteq \text{coNP/poly}$ .

## 4.3 Polynomial Kernels by Combining Budget with Structural Graph Parameters

Finally, we investigate colored cut games from the viewpoint of kernelization. By the above, natural parameterizations by  $b(I)$  or even  $|C|$  will not give a kernel. Moreover, COLORED  $(s, t)$ -CUT is NP-hard even if the vertex cover number of the input graph is at most two [28]. Hence, for most structural graph parameters there is little hope to obtain polynomial kernels. We will show that, however, all colored cut games admit polynomial kernels when parameterized by the total attacker budget  $k$  and the vertex cover number. In fact, we show polynomial kernels for smaller parameters. To this end, we consider generalizations of vertex covers.

► **Definition 4.6.** For a graph  $G$ , we let  $\text{lp}(G)$  denote the length of a longest path in  $G$ . We call a vertex set  $S \subseteq V$  an  $r$ -lp-modulator in  $G$  if  $\text{lp}(G - S) \leq r$ . The size of a smallest  $r$ -lp-modulator of a graph  $G$  is the  $r$ -lp-deletion number  $\kappa_r$  of  $G$ .

Thus, an  $r$ -lp-modulator is a vertex set whose deletion results in a graph that has no simple paths of length at least  $r + 1$ . Clearly, the  $r$ -lp-deletion number of  $G$  is monotonically decreasing with  $r$ . Note that the vertex cover number is exactly the 1-lp-deletion number. More generally, if every connected component of a graph has order at most  $r$ , then  $\text{lp}(G) \leq r$ . Thus, the  $r$ -lp-deletion number of a graph is never larger than the so-called  $r$ -COC number, the smallest size of a vertex set whose deletion results in a graph where every connected component has order at most  $r$ .

To show the correctness of the kernelization, we need to argue that an attacker can achieve a colored cut in the kernel if and only if he can achieve it in the input instance. Thus, we only need to consider colored cuts of bounded size in the correctness proof. Motivated by this, we generalize the notion of colored-cut-equivalence as follows.

► **Definition 4.7.** Let  $x$  be an integer. Two colored graphs  $\mathcal{H} = (G, s, t, C, \ell)$  and  $\mathcal{H}' = (G', s', t', C, \ell')$  are  $x$ -colored-cut-equivalent if for all  $\tilde{C} \subseteq C$  of size at most  $x$  it holds that  $\tilde{C}$  is a colored  $(s, t)$ -cut in  $G$  if and only if  $\tilde{C}$  is a colored  $(s', t')$ -cut in  $G'$ .

Since the total attacker budget is an upper bound for the size of the colored  $(s, t)$ -cut the attacker can choose, we obtain the following.

► **Corollary 4.8.** *Two instances  $I = (\mathcal{H}, \vec{d}, \vec{a})$  and  $I' = (\mathcal{H}', \vec{d}, \vec{a})$  of any colored cut game are equivalent if  $\mathcal{H}$  and  $\mathcal{H}'$  are  $k$ -colored-cut-equivalent where  $k = \sum_{x=1}^i a_x$ .*

Now, we show that we can compute in polynomial time a  $k$ -colored-cut-equivalent graph which  $(k + \kappa_r)^{\mathcal{O}(r)}$  edges.

► **Lemma 4.9.** *Let  $\mathcal{H} = (G = (V, E), s, t, C, \ell)$  be a colored graph with  $r$ -lp-deletion number  $\kappa_r$  and let  $k \leq |C|$  be an integer. Then, one can compute in  $|\mathcal{H}|^{\mathcal{O}(r)}$  time a  $k$ -colored-cut-equivalent graph  $\mathcal{H}' = (G' = (V', E'), s', t', C, \ell')$  with at most  $\binom{(r+1)\kappa_r+2}{2} \cdot (r+1)(r+1)!k^{r+1}$  edges.*

The idea of the algorithm is the following: First, we approximate an  $r$ -lp-modulator  $\Gamma$  containing both  $s$  and  $t$  and compute for each pair  $\{x, y\}$  of vertices of  $\Gamma$  the collection  $A_{\{x, y\}}$  of all color sets of  $(x, y)$ -paths not containing other vertices of  $\Gamma$ . For each such pair, we compute the HITTING SET-instance  $(A_{\{x, y\}}, k)$  and kernelize it to a HITTING SET-instance  $(A'_{\{x, y\}}, k)$  with  $|A'_{\{x, y\}}| < (r+1)!k^{r+1}$  by using the Sunflower Lemma [13]. Finally, we construct a colored graph  $\mathcal{H}'$  such that  $\Gamma$  is an  $r$ -lp-modulator of  $G'$  and such that for each pair  $\{x, y\}$  of vertices of  $\Gamma$ , the collection of all color sets of  $(x, y)$ -paths not containing other vertices of  $\Gamma$  is precisely  $A'_{\{x, y\}}$ . This can be done with  $|A'_{\{x, y\}}|$  paths for each  $A'_{\{x, y\}}$ . Hence, the resulting graph has bounded size.

We now describe in detail how to construct  $\mathcal{H}'$ . First, we compute an  $r$ -lp-modulator  $\Gamma$  of size at most  $\kappa_r(r+1)+2$  containing  $s$  and  $t$  via the following  $(r+1)$ -approximation algorithm: Start with an empty set  $\Gamma'$ . While the graph  $G - \Gamma'$  contains a path of length at least  $r+1$ , add the  $r+1$  vertices of this path to  $\Gamma'$ . Afterwards, we set  $\Gamma := \Gamma' \cup \{s, t\}$ . By construction,  $\Gamma$  is an  $r$ -lp-modulator and it has size at most  $\kappa_r(r+1)+2$  since every  $r$ -lp-modulator contains at least one vertex of each path of length at least  $r+1$ .

Since  $G - \Gamma$  has no paths of length at least  $r+1$ , we know that every path between two vertices of  $\Gamma$ , which does not contain a third vertex of  $\Gamma$ , has at most  $r+1$  edges. We compute for every  $\{a, b\} \in \binom{\Gamma}{2}$  the family of all color sets  $A_{\{a, b\}}$  of  $(a, b)$ -paths in  $G_{\{a, b\}} := G - (\Gamma \setminus \{a, b\})$ . That is,  $A_{\{a, b\}} = \mathcal{C}(\mathcal{H}_{\{a, b\}})$ , where  $\mathcal{H}_{\{a, b\}} := (G_{\{a, b\}}, a, b, C, \ell)$ . Hence, for every color set  $\tilde{C} \subseteq C$  it holds that  $\tilde{C}$  is a colored  $(a, b)$ -cut in  $G_{\{a, b\}}$  if and only if  $\tilde{C}$  is a hitting set for  $A_{\{a, b\}}$ . Note that  $A_{\{a, b\}}$  contains only color sets of size at most  $r+1$ . Next, we reduce each of the sets  $A_{\{a, b\}}$  to a size of at most  $(r+1)! \cdot k^{r+1}$  using a well known reduction rule for  $(r+1)$ -HITTING SET. This reduction rule uses the famous Sunflower Lemma [13].

► **Lemma 4.10.** *If  $A_{\{a, b\}}$  has size more than  $(r+1)! \cdot k^{r+1}$ , then there are  $k+1$  distinct sets  $S_1, \dots, S_{k+1} \in A_{\{a, b\}}$  that can be computed in polynomial time such that  $S_j \cap S_{j'} = \bigcap_{1 \leq i \leq k+1} S_i =: \mathcal{S}$  for all distinct  $j, j' \in [1, k+1]$ .*

► **Rule 4.1.** *If  $|A_{\{a, b\}}| > (r+1)! \cdot k^{r+1}$ , then compute sets  $S_1, \dots, S_{k+1} \in A_{\{a, b\}}$  and  $\mathcal{S}$  with the property of Lemma 4.10.*

- If  $\mathcal{S} = \emptyset$ , then remove all sets of  $A_{\{a, b\}}$  except  $\{S_1, \dots, S_{k+1}\}$ .
- Otherwise, remove  $S_1, \dots, S_{k+1}$  from  $A_{\{a, b\}}$  and add the set  $\mathcal{S}$ .

Next, we show that the rule is correct in the following sense.

► **Proposition 4.11.** *Let  $\tilde{C} \subseteq C$  be a set of size at most  $k$ .*

- If  $\mathcal{S} \neq \emptyset$ , then  $\tilde{C}$  is a hitting set for  $A_{\{a, b\}}$  if and only if  $\tilde{C}$  is a hitting set for  $\{\mathcal{S}\} \cup (A_{\{a, b\}} \setminus \{S_i \mid 1 \leq i \leq k+1\})$ .
- If  $\mathcal{S} = \emptyset$ , then  $\tilde{C}$  is a hitting set for  $A_{\{a, b\}}$  if and only if  $\tilde{C}$  is a hitting set for  $\{S_i \mid 1 \leq i \leq k+1\}$ .

Let  $A'_{\{a,b\}}$  be the set obtained after exhaustively applying Rule 4.1 to  $A_{\{a,b\}}$ . By the definition of Rule 4.1,  $A'_{\{a,b\}}$  has size at most  $(r+1)! \cdot k^{r+1}$ . Moreover, by the definition of  $A_{\{a,b\}}$  and Proposition 4.11, we obtain that every color set  $\tilde{C} \subseteq C$  of size at most  $k$  is a colored  $(a,b)$ -cut in  $G_{\{a,b\}}$  if and only if  $\tilde{C}$  is a hitting set for  $A'_{\{a,b\}}$ .

Finally, we define the colored graph  $\mathcal{H}'$ . We start with a graph  $G'$  containing only the vertices of  $\Gamma$  and set  $s' = s$  and  $t' = t$ . Next, for every set  $\{a,b\} \in \binom{\Gamma}{2}$  and every color set  $L \in A'_{\{a,b\}}$ , we add an  $(a,b)$ -path  $P_L$  with  $\max(1, |L| - 1)$  new internal vertices to  $G'$  and color the edges of  $P$  in such a way that  $\ell'(P'_L) := L$ , where  $P'_L := a \cdot P_L \cdot b$ . This finishes the definition of  $\mathcal{H}'$ . We may now show the correctness and the running time of the data reduction and the size bound of the resulting graph  $\mathcal{H}'$ .

**Proof of Lemma 4.9.** Note that  $\mathcal{C}(\mathcal{H}'_{\{a,b\}}) = A'_{\{a,b\}}$ , where  $G'_{\{a,b\}} := G' - (\Gamma \setminus \{a,b\})$  and  $\mathcal{H}'_{\{a,b\}} := (G'_{\{a,b\}}, a, b, C, \ell')$ . Hence, we obtain that every color set  $\tilde{C} \subseteq C$  of size at most  $k$  is a colored  $(a,b)$ -cut in  $G'_{\{a,b\}}$  if and only if  $\tilde{C}$  is a hitting set for  $A'_{\{a,b\}}$ . By the above, this is the case if and only if  $\tilde{C}$  is a colored  $(a,b)$ -cut in  $G_{\{a,b\}}$ . Consequently,  $\mathcal{H}_{\{a,b\}}$  and  $\mathcal{H}'_{\{a,b\}}$  are  $k$ -colored-cut-equivalent.

Now, we use this fact to prove that  $\mathcal{H}$  and  $\mathcal{H}'$  are  $k$ -colored-cut-equivalent. Let  $\tilde{C}$  be a colored  $(s,t)$ -cut of size at most  $k$  in  $G$ . We show that  $\tilde{C}$  is a colored  $(s,t)$ -cut in  $G'$ . Assume towards a contradiction, that this is not the case. Then, there is an  $(s,t)$ -path  $P' = (u_1, \dots, u_q)$  in  $G'$  with  $u_1 = s$  and  $u_q = t$  such that  $\ell'(P') \cap \tilde{C} = \emptyset$ . Let  $u_{i_1}, \dots, u_{i_z}$  be the vertices of  $\Gamma$  in  $P'$  in the ordering of the traversal of the path. Recall that  $s \in \Gamma$  and  $t \in \Gamma$ , which implies that  $u_{i_1} = u_1$  and  $u_{i_z} = u_q$ . Now, let  $P'_j := (u_{i_j}, u_{i_{j+1}}, \dots, u_{i_{(j+1)}-1}, u_{i_{(j+1)}})$  denote the subpath of  $P'$  connecting  $u_{i_j}$  and  $u_{i_{j+1}}$  for all  $j \in [1, z-1]$ . Due to the fact that  $\ell'(P') \cap \tilde{C} = \emptyset$ , it follows that  $\ell'(P'_j) \cap \tilde{C} = \emptyset$  for all  $j \in [1, z-1]$ . Thus, for each  $j \in [1, z-1]$  it holds that  $\tilde{C}$  is not a colored  $(u_{i_j}, u_{i_{j+1}})$ -cut in  $G'_{\{u_{i_j}, u_{i_{j+1}}\}}$ . Moreover, since for each  $j \in [1, z-1]$ ,  $\mathcal{H}_{\{u_{i_j}, u_{i_{j+1}}\}}$  and  $\mathcal{H}'_{\{u_{i_j}, u_{i_{j+1}}\}}$  are  $k$ -colored-cut-equivalent, it follows that there is an  $(u_{i_j}, u_{i_{j+1}})$ -path  $P_j$  in  $G_{\{u_{i_j}, u_{i_{j+1}}\}}$  such that  $\ell(P_j) \cap \tilde{C} = \emptyset$ . By connecting all paths  $P_1 \rightsquigarrow \dots \rightsquigarrow P_{z-1}$ , we get an  $(s,t)$ -path  $P$  in  $G$  with  $\ell(P) \cap \tilde{C} = \bigcup_{j=1}^{z-1} (\ell(P_j) \cap \tilde{C}) = \emptyset$ . This contradicts the assumption that  $\tilde{C}$  is a colored  $(s,t)$ -cut in  $G$ . The opposite direction can be shown analogously.

Next, we show the running time of the construction. Since paths of length at least  $r+1$  can be computed in  $2^{\mathcal{O}(r)} \cdot |V|^{\mathcal{O}(1)}$  time [1], we can compute the set  $\Gamma$  in the same running time. Moreover, since no  $(a,b)$ -path in  $G_{\{a,b\}}$  has length more than  $r+2$ , we can compute all the sets  $A_{\{a,b\}}$  in  $\mathcal{O}(\binom{|\Gamma|}{2} \cdot |V|^{r+\mathcal{O}(1)})$  time. Since each application of Rule 4.1 takes only polynomial time and reduces the size of  $A_{\{a,b\}}$  by at least one, all the sets  $A'_{\{a,b\}}$  can be computed in  $\mathcal{O}(\binom{|\Gamma|}{2} \cdot |V|^{r+\mathcal{O}(1)})$  time as well. Thus, the complete construction takes  $\mathcal{O}(\binom{|\Gamma|}{2} \cdot 2^{\mathcal{O}(r)} \cdot |V|^{r+\mathcal{O}(1)})$  time.

Finally, we show the size of the kernel. By construction,  $G'$  contains for every  $\{a,b\} \in \binom{\Gamma}{2}$  at most  $|A'_{\{a,b\}}| \leq (r+1)!k^{r+1}$  paths with at most  $r+1$  edges each. Consequently,  $G'$  contains at most  $\binom{|\Gamma|}{2} \cdot (r+1)(r+1)!k^{r+1}$  edges. Since  $|\Gamma|$  has size at most  $(r+1)\kappa_r + 2$ , we obtain the stated kernel size. ◀

Corollary 4.8 and Lemma 4.9 lead to the following kernelization.

► **Theorem 4.12.** *For each constant  $r \geq 1$ , every colored cut game admits a polynomial kernel with at most  $\binom{(r+1)\kappa_r + 2}{2} \cdot (r+1)(r+1)!k^{r+1}$  edges when parameterized by the  $r$ -lp-deletion number  $\kappa_r$  of  $G$  and the total attacker budget  $k$ .*

► **Corollary 4.13.** *Every colored cut game admits a polynomial kernel with at most  $\binom{2vc+2}{2} \cdot 4k^2$  edges when parameterized by the vertex cover number  $vc$  of  $G$  and the total attacker budget  $k$ .*

A further parameter to consider in this context is the treedepth of  $G$  [23]: The treedepth  $td(G)$  of a graph is at least  $\log(lp(G))$  [23]. Thus, Theorem 4.12 also implies the following result for modulators to graphs with treedepth at most  $r$ . Herein  $\lambda_r$  denotes the size of a smallest treedepth  $r$ -modulator.

► **Corollary 4.14.** *For any constant  $r \geq 1$ , every colored cut game admits a polynomial kernel when parameterized by the size  $\lambda_r$  of a smallest treedepth  $r$ -modulator and the total attacker budget  $k$ .*

The size of the kernel is  $(\lambda_r)^2 k^{\mathcal{O}(2^r)}$  and thus the guarantee is not of practical interest even for rather moderate values of  $k$  and the treedepth bound  $r$ . However, these kernelization results are optimal in the following two ways: First, COLORED  $(s, t)$ -CUT does not admit a kernel with respect to  $k$  even on graphs with treewidth two [15]. Hence, we may not replace  $r$ -lp-modulators or treedepth- $r$  modulators by treewidth- $r$  modulators. Moreover, the standard reduction from  $(r + 1)$ -HITTING SET to COLORED  $(s, t)$ -CUT gives graphs in which  $s$  and  $t$  are connected only via vertex disjoint paths of length at most  $r + 2$ . Hence,  $lp(G - \{s, t\}) \leq r$  and, thus,  $\kappa_r \leq 2$ . Moreover,  $k$  is exactly the budget of the HITTING SET instance. Thus, since  $(r + 1)$ -HITTING SET does not admit a compression of bitsize  $k^{r+1-\epsilon}$  unless  $NP \subseteq coNP/poly$  [10], COLORED  $(s, t)$ -CUT does not admit a kernel of size  $k^{r+1-\epsilon}$  even if it has a  $r$ -lp-deletion number of size two. Since in these simple graphs produced by the reduction, we have  $td(G) \in \Theta(\log lp(G))$ , we can also not improve on the doubly exponential dependence on  $r$  in the exponent of the kernelization for treedepth.

Based on these kernel results, it also follows that all colored cut games admit FPT-algorithms when parameterized by  $\kappa_r + k$ . In the following, we describe FPT-algorithms for COLORED  $(s, t)$ -CUT and DA-CCV when parameterized by  $\kappa_r + k$  with a better running time than a simple brute-force on the kernel.

► **Theorem 4.15.** *For any constant  $r \geq 1$ , COLORED  $(s, t)$ -CUT can be solved in  $(2^{\kappa_r}(r + 1))^k + (r + 1)^{\kappa_r} \cdot n^{\mathcal{O}(r)}$  time, where  $\kappa_r$  denotes the  $r$ -lp-deletion number of  $G$  and  $k$  denotes the budget of the attacker.*

**Proof.** First, we compute an  $r$ -lp-modulator  $\Gamma'$  of size  $\kappa_r$  in  $(r + 1)^{\kappa_r} n^{\mathcal{O}(r)}$  time using a search tree algorithm that checks whether a graph contains a simple path of length  $r + 1$  and branches on the possibilities to destroy this path via vertex deletion. Afterwards, we check for each of the  $2^{\kappa_r}$  many partitions  $(S, T)$  of  $\Gamma := \Gamma' \cup \{s, t\}$  with  $s \in S$  and  $t \in T$ , if there is a color set  $\tilde{C} \subseteq C$  of size at most  $k$  such that there is no connected component containing both a vertex of  $S$  and a vertex of  $T$  after removing all the edges colored in  $\tilde{C}$ . To this end, we first compute for every pair of vertices  $x \in S$  and  $y \in T$  the collection  $A_{\{x, y\}}$  of all color sets of  $(x, y)$ -paths in  $G_{\{x, y\}} := G - (\Gamma \setminus \{x, y\})$ . This can be done in  $n^{\mathcal{O}(r)}$  time since  $G_{\{x, y\}}$  does not contain any  $(x, y)$ -path of length more than  $r + 2$ . To check if there is a color set  $\tilde{C} \subseteq C$  of size at most  $k$  with the intended property, we only have to check if  $\tilde{C} \cap L \neq \emptyset$  for all pairs of vertices  $x \in S$  and  $y \in T$  and all  $L \in A_{\{x, y\}}$ . This is equivalent to the question, if there is a hitting set of size at most  $k$  for  $\bigcup_{(x, y) \in S \times T} A_{\{x, y\}}$ , which can be determined in  $(r + 1)^k n^{\mathcal{O}(1)}$  time due to the fact that every set  $A_{\{x, y\}}$  contains only color sets of size at most  $r + 1$  and  $(r + 1)$ -HITTING SET can be solved in  $(r + 1)^k n^{\mathcal{O}(1)}$  time. ◀

► **Corollary 4.16.** *COLORED  $(s, t)$ -CUT can be solved in  $2^{vc+k} n^{\mathcal{O}(1)}$  time, where  $vc$  denotes the vertex cover number of  $G$  and  $k$  denotes the budget of the attacker.*

■ **Table 1** Classic Complexity of COLORED  $(s, t)$ -CUT,  $(DA)^i$ -CCR,  $A(DA)^i$ -CCR, and  $(DA)^*$ -CCR in general and in some restricted cases.

graph classes	COLORED $(s, t)$ -CUT	$(DA)^i$ -CCR	$A(DA)^i$ -CCR	$(DA)^*$ -CCR
general	NP-c [7, 15]	$\Sigma_{2i}^P$ -c	$\Pi_{2i+1}^P$ -c	PSPACE-c
subcubic	$\in P$	$\Sigma_{2i}^P$ -c	$\Sigma_{2i}^P$ -h	PSPACE-c
bipartite planar	NP-c [28]	$\Sigma_{2i}^P$ -c	$\Pi_{2i+1}^P$ -c	PSPACE-c
bipartite planar subcubic	$\in P$	$\Sigma_{2i}^P$ -c	$\Sigma_{2i}^P$ -h	PSPACE-c
every color in $\leq 2$ $(s, t)$ -paths	$\in P$ [27]	NP-h NP-c if $i = 1$	NP-h	NP-h

We extend our fixed-parameter tractability result from COLORED  $(s, t)$ -CUT to  $(DA)^1$ -CCV.

► **Theorem 4.17.** *For any constant  $r \geq 1$ ,  $(DA)^1$ -CCV can be solved in  $((2k)^{\kappa_r} (r+1)^k + (r+1)^{\kappa_r}) \cdot n^{\mathcal{O}(r)}$  time, where  $\kappa_r$  denotes the  $r$ -lp-deletion number of  $G$  and  $k$  denotes the budget of the attacker.*

Let us remark that it would also be natural to attempt to generalize the vertex cover number to the vertex deletion distance to a maximum degree of  $r$  for any  $r \in \mathbb{N}$ . Note, however, that the standard reduction from HITTING SET to COLORED  $(s, t)$ -CUT [7] already implies that COLORED  $(s, t)$ -CUT has no kernel of size  $|C|^{\mathcal{O}(1)}$  even when  $G$  has only two vertices of degree at least three, unless  $NP \subseteq coNP/poly$ . Hence, for any  $r \geq 2$  it is unlikely that we can obtain polynomial kernels for  $|C|$  plus the vertex deletion distance to a maximum degree of  $r$ .

## 5 Conclusion

We have studied the complexity of a variety of games that deal with preventing or establishing a colored cut in edge-colored graphs (see Table 1 for an overview of the classic complexity results). In the negative and the positive results of this work we exploited the close connection between colored cut games and the HITTING SET problem. For example, the PSPACE-hardness proof for the most general game presented in this work, is based on a simple reduction from COMPETITIVE HITTING SET. Ideally, we would have liked to also use such a simple reduction for the games with a constant number of rounds. However, we do not know whether the corresponding HITTING SET games are hard. In particular, it seems open whether the following problem is  $\Pi_2^P$ -hard.

$\forall \exists$  HITTING SET

**Input:** A collection  $\mathcal{F}$  of subsets of a universe  $\mathcal{U}$  and two integers  $k_1$  and  $k_2$ .

**Question:**  $\forall D \in \binom{\mathcal{U}}{k_1}, \exists A \in \binom{\mathcal{U} \setminus D}{k_2}$  such that  $A \cap F \neq \emptyset$  for all  $F \in \mathcal{F}$

This problem asks for a winning strategy for the attacker who wants to complete a hitting set in the case that the defender starts. If this problem is  $\Pi_2^P$ -hard, then we can infer the  $\Pi_2^P$ -hardness of  $(DA)^1$ -CCV directly from it. Otherwise, the hardness of  $(DA)^1$ -CCV would be rooted in the fact that we can create an exponential number of paths in our hardness construction. It would also be interesting to explore further how efficiently we can reduce from colored cut games to HITTING SET. In other words, how long does it take to construct  $\mathcal{C}(\mathcal{H})$ , the collection of color sets of  $(s, t)$ -paths, for a given colored graph  $\mathcal{H}$ ? In particular, can we compute the set  $\mathcal{C}(\mathcal{H})$  in  $|\mathcal{C}(\mathcal{H})| \cdot |\mathcal{H}|^{\mathcal{O}(1)}$  time?

---

References

---

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
- 2 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- 3 Mordechai Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition*. Springer, 2012.
- 4 Augusto Bordini, Fábio Protti, Thiago Gouveia da Silva, and Gilberto Farias de Sousa Filho. New algorithms for the minimum coloring cut problem. *International Transactions in Operational Research*, 26(5):1868–1883, 2019.
- 5 John Bruno and Louis Weinberg. A constructive graph-theoretic solution of the Shannon switching game. *IEEE Transactions on Circuit Theory*, 17(1):74–81, 1970.
- 6 Stephen M. Chase. An implemented graph algorithm for winning Shannon switching games. *Communications of the ACM*, 15(4):253–256, 1972.
- 7 David Coudert, Pallab Datta, Stephane Perennes, Hervé Rivano, and Marie-Emilie Voge. Shared risk resource group complexity and approximability issues. *Parallel Processing Letters*, 17(2):169–184, 2007.
- 8 David Coudert, Stéphane Pérennes, Hervé Rivano, and Marie-Emilie Voge. Combinatorial optimization in networks with shared risk link groups. *Discrete Mathematics & Theoretical Computer Science*, 18(3), 2016.
- 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 10 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014.
- 11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 12 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
- 13 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.
- 14 Seyed Rasoul Etesami and Tamer Basar. Dynamic games in cyber-physical security: An overview. *Dynamic Games and Applications*, 9(4):884–913, 2019.
- 15 Michael R. Fellows, Jiong Guo, and Iyad A. Kanj. The parameterized complexity of some minimum label problems. *Journal of Computer and System Sciences*, 76(8):727–740, 2010.
- 16 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- 17 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- 18 Sulamita Klein, Luerbio Faria, Ignasi Sau, Rubens Sucupira, and Uéverton Souza. On colored edge cuts in graphs. In *Proceedings of the 1st Encontro de Teoria da Computação (ETC '16)*, Sociedade Brasileira de Computação, pages 780–783. CSBC, 2016.
- 19 Xiannuan Liang and Yang Xiao. Game theory for network security. *IEEE Communications Surveys & Tutorials*, 15(1):472–486, 2013.
- 20 Jelena Mirkovic, Peter Reiher, Christos Papadopoulos, Alefiya Hussain, Marla Shepard, Michael Berg, and Robert Jung. Testing a collaborative ddos defense in a red team/blue team exercise. *IEEE Transactions on Computers*, 57(8):1098–1112, 2008.
- 21 Nils Morawietz. Computational complexity of network robustness in edge-colored graphs. Master’s thesis, Philipps-Universität Marburg, 2019.
- 22 Nils Morawietz, Niels Grüttemeier, Christian Komusiewicz, and Frank Sommer. Refined parameterizations for computing colored cuts in edge-colored graphs. In *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Informatics, (SOFSEM '20)*, volume 12011 of *LNCS*, pages 248–259. Springer, 2020.

- 23 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.
- 24 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- 25 Sankardas Roy, Charles Ellis, Sajjan G. Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *Proceedings of the 43rd Hawaii International International Conference on Systems Science (HICSS '10)*, pages 1–10. IEEE Computer Society, 2010.
- 26 Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978.
- 27 Rubens André Sucupira. *Problemas de cortes de arestas máximos e mínimos em grafos*. PhD thesis, Universidade Federal do Rio de Janeiro, 2017.
- 28 Yongge Wang and Yvo Desmedt. Edge-colored graphs with applications to homogeneous faults. *Information Processing Letters*, 111(13):634–641, 2011.
- 29 Shengli Yuan, Saket Varma, and Jason P. Jue. Minimum-color path problems for reliability in mesh networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05)*, pages 2658–2669, 2005.
- 30 Rico Zenklusen. Matching interdiction. *Discrete Applied Mathematics*, 158(15):1676–1690, 2010.
- 31 Peng Zhang and Bin Fu. The label cut problem with respect to path length and label frequency. *Theoretical Computer Science*, 648:72–83, 2016.



# Randomness Efficient Noise Stability and Generalized Small Bias Sets

**Dana Moshkovitz**

University of Texas at Austin, Department of Computer Science, TX, USA  
danama@cs.utexas.edu

**Justin Oh**

University of Texas at Austin, Department of Computer Science, TX, USA  
sjo@cs.utexas.edu

**David Zuckerman**

University of Texas at Austin, Department of Computer Science, TX, USA  
diz@cs.utexas.edu

---

## Abstract

We present a randomness efficient version of the linear noise operator  $T_\rho$  from boolean function analysis by constructing a sparse linear operator on the space of boolean functions  $\{0, 1\}^n \rightarrow \{0, 1\}$  with similar eigenvalue profile to  $T_\rho$ . The linear operator we construct is a direct consequence of a generalization of  $\epsilon$ -biased sets to the product distribution  $\mathcal{D}_p$  on  $\{0, 1\}^n$  where the marginal of each coordinate is  $p = \frac{1}{2} - \frac{1}{2}\rho$ . Such a generalization is a small support distribution that fools linear tests when the input of the test comes from  $\mathcal{D}_p$  instead of the uniform distribution. We give an explicit construction of such a distribution that requires  $\log n + O_p(\log \log n + \log \frac{1}{\epsilon})$  bits of uniform randomness to sample from, where the  $p$  subscript hides  $O(\log^2 \frac{1}{p})$  factors. When  $p$  and  $\epsilon$  are constant, this yields a support size nearly linear in  $n$ , whereas previous best known constructions only guarantee a size of  $\text{poly}(n)$ . Furthermore, our construction implies an explicitly constructible “sparse” noisy hypercube graph that is a small set expander.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pseudorandomness and derandomization

**Keywords and phrases** pseudorandomness, derandomization, epsilon biased sets, noise stability

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.31

**Funding** *Dana Moshkovitz*: Supported in part by NSF Grant CCF-1705028 and CCF-1648712.

*Justin Oh*: Supported by NSF Grant CCF-1705028.

*David Zuckerman*: Supported in part by NSF Grant CCF-1705028 and a Simons Investigator Award (#409864).

## 1 Introduction

Most constructions in pseudorandomness aim to simulate the behavior of a class of tests when the input to the tests are drawn from the *uniform* distribution on  $\{0, 1\}^n$ . Simulating the uniform distribution has been the main subject of attention because many algorithmic problems ultimately boil down to finding a solution to a problem in an input space where a large fraction of inputs are correct. Thus a uniform sample from the space will find a correct solution with high probability. However, other distributions have also proved to be incredibly useful in solving important problems in computer science. One example of such a distribution is the product distribution with marginals  $p$ :

► **Definition 1** (product distribution with marginals  $p$ ). *Let  $p \in [0, 1]$ . The product distribution with marginals  $p$  is the distribution  $\mathcal{D}_{p,n}$  on  $\{0, 1\}^n$  where each bit  $x_i$  is picked independently with  $\Pr[x_i = 1] = p$ . When the length of the string  $n$  is clear from context we simply denote the distribution as  $\mathcal{D}_p$ .*



© Dana Moshkovitz, Justin Oh, and David Zuckerman;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 31; pp. 31:1–31:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Apart from being one of the simplest deviations from the uniform distribution,  $\mathcal{D}_p$  in particular serves an integral role in the concept of the noise stability of boolean functions. Noise stability is a fundamental concept in boolean function analysis that is pervasive in many branches of mathematics such as social choice theory [13], and has a crucial application in celebrated results in hardness of approximation [7, 8]. Roughly speaking, the stability of a boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is a measure of how likely the output is to change when each input bit is independently flipped with some small probability  $p$ . The bit flipping is generally thought of as noise, where input  $\mathbf{x} \in \{0, 1\}^n$  is perturbed to  $\mathbf{x} + \boldsymbol{\mu}$  for  $\boldsymbol{\mu} \sim \mathcal{D}_p$ . If we instead draw  $\mathbf{z} \sim \mathcal{Z}$  and perturb  $\mathbf{x}$  to  $\mathbf{x} + \mathbf{z}$  for  $\mathcal{Z}$  that is a randomness efficient approximation of  $\mathcal{D}_p$  (under the right notion of approximation), we can then define a randomness efficient notion of noise. In addition to suggesting a randomness efficient noise test, we believe that the existence of such a notion of noise is of independent interest.

An alternative view of the concept of noise stability relates to the *noise operator*  $T_p$ ,<sup>1</sup> which is a linear operator that acts on truth tables of functions  $\mathbf{f} : \{-1, 1\}^n \rightarrow \{-1, 1\}$ . The matrix corresponding to  $T_p$  is simply the  $2^n \times 2^n$  transition matrix of the graph on  $\{0, 1\}^n$  where a random step from  $\mathbf{x}$  moves to  $\mathbf{x} + \mathbf{n}$  for  $\mathbf{n} \sim \mathcal{D}_{p,n}$ . Many important properties of the noise operator and noise stability stem from the eigenvalues of  $T_p$ . Thus we focus on defining a linear noise operator with similar eigenvalue profile to  $T_p$ . We show that in order to do so it suffices to study a generalization of  $\epsilon$ -biased sets.

Small bias sets are a fundamental object in pseudorandomness, with applications to error-correcting codes, derandomization, and PCPs [12, 15, 4]. An  $\epsilon$ -biased set is a small subset  $S \subset \{0, 1\}^n$  such that a uniform random sample from  $S$  behaves similarly to a uniform random sample from all of  $\{0, 1\}^n$  with respect to linear tests. More formally,  $S$  is an  $\epsilon$ -biased set if for any nonempty subset of indices  $I \subset [n]$ , the *bias* of  $I$  is small: if  $\mathcal{U}(S)$  is the uniform distribution on  $S$  then:

$$\left| Pr_{\mathbf{x} \sim \mathcal{U}(S)} \left( \bigoplus_{i \in I} x_i = 0 \right) - Pr_{\mathbf{x} \sim \mathcal{U}(S)} \left( \bigoplus_{i \in I} x_i = 1 \right) \right| \leq \epsilon$$

In other words, the parity of any subset of indices has almost equal probability of being 0 or 1. Notice that in the case of a uniform random sample over  $\{0, 1\}^n$ , the parity of any nonempty subset is equally likely to be 0 or 1. Hence  $\epsilon$ -biased sample spaces approximate the uniform distribution in the sense that parities of subsets of indices behave almost the way they should. Classic results show that there are  $\epsilon$ -biased sets that require  $O(\log \frac{n}{\epsilon})$  bits of uniform randomness to sample from. In other words there are explicit constructions where the size of  $S$  is polynomial in  $n$ , and optimal constructions even have size linear in  $n$  [12, 15]. In addition to having applications in randomness efficient noise, it is a natural question to ask whether there are small sample spaces that approximate distributions on  $\{0, 1\}^n$  other than the uniform distribution.

## 1.1 Our Contribution

We generalize  $\epsilon$ -biased sets for the distribution  $\mathcal{D}_p$  on  $\{0, 1\}^n$ . The sample space  $\mathcal{Z}$  we construct approximates  $\mathcal{D}_p$  in the sense that if  $\mathbf{z} \sim \mathcal{Z}$  then for every  $I \subset [n]$  the parity of  $\mathbf{z}_I$  has approximately the same distribution as when  $\mathbf{z}$  is drawn from  $\mathcal{D}_p$ .

<sup>1</sup> In mainstream literature, the noise operator that we denote  $T_p$  is instead denoted as  $T_\rho$  for  $\rho = 1 - 2p$ . We stray from the standard notation in this paper for convenience with our own notation

► **Theorem 2** (Main Result). *Let  $p$  be a power of 2. There exists a distribution  $\mathcal{Z}$  on  $\{0, 1\}^n$  such that for every  $I \subset [n]$  we have:*

$$\left| \Pr_{\mathbf{z} \sim \mathcal{Z}} \left( \bigoplus_{i \in I} z_i = 1 \right) - \Pr_{\mathbf{r} \sim \mathcal{D}_{p,n}} \left( \bigoplus_{i \in I} r_i = 1 \right) \right| \leq \epsilon$$

$\mathcal{Z}$  requires  $\log n + O(\log \frac{1}{p} \log \log n + \log^2 \frac{1}{p} + \log \frac{1}{p} \log \frac{1}{\epsilon})$  bits of uniform randomness to sample from. Moreover, the support of  $\mathcal{Z}$  (along with the corresponding probability of each point) can be explicitly constructed in time  $n \cdot \text{poly}(\log n, \frac{1}{\epsilon})$  for constant  $p$ .

The main takeaway from our result is that there is a simple *explicit* construction of a distribution that approximates  $\mathcal{D}_{p,n}$  with support size *nearly linear* in  $n$  when  $p$  and  $\epsilon$  are constant. This roughly matches the size of an optimal  $\epsilon$ -biased set, although the size blows up for nonconstant  $p$ .

## 1.2 Application to Randomness Efficient Noise

The main application of our generalization of  $\epsilon$ -biased sets is in the definition of a “randomness efficient” version of noise stability. The stability of the function  $\mathbf{f}$  is defined as:

$$\text{Stab}_{1-2p}(\mathbf{f}) = \langle \mathbf{f}, T_p \mathbf{f} \rangle$$

Our construction of  $\epsilon$ -biased sets for  $\mathcal{D}_{p,n}$  naturally suggests a new noise operator  $T_{p,\epsilon}^{\text{sparse}}$  that is the transition matrix of the graph where a random step from  $\mathbf{x}$  moves to  $\mathbf{x} + \mathbf{z}$  for  $\mathbf{z}$  a sample from our constructed distribution  $\mathcal{Z}$ . We can then define a new notion of stability:

$$\text{Stab}_{1-2p}^{\text{sparse}}(\mathbf{f}) = \langle \mathbf{f}, T_{p,\epsilon}^{\text{sparse}} \mathbf{f} \rangle$$

Through analysis of the eigenvalues of  $T_p$  and  $T_{p,\epsilon}^{\text{sparse}}$ , we can show that our new notion of stability is the same as the original up to an additive error of  $2\epsilon$ :

► **Theorem 3** (Randomness Efficient Approximate Noise Stability). *Let  $\mathbf{f} : \{-1, 1\}^n \rightarrow [0, 1]$ . Let  $\text{Stab}_{1-2p}(\mathbf{f}) = \langle \mathbf{f}, T_p \mathbf{f} \rangle$  be the stability of  $f$  under the noise operator  $T_p$ . Let  $\text{Stab}_{1-2p}^{\text{sparse}}(\mathbf{f}) = \langle \mathbf{f}, T_{p,\epsilon}^{\text{sparse}} \mathbf{f} \rangle$  be the stability of  $\mathbf{f}$  under the noise operator  $T_{p,\epsilon}^{\text{sparse}}$  defined by our  $\epsilon$ -biased set for  $\mathcal{D}_p$ . Then:*

$$|\text{Stab}_{1-2p}(\mathbf{f}) - \text{Stab}_{1-2p}^{\text{sparse}}(\mathbf{f})| \leq 2\epsilon$$

An immediate consequence of the above theorem is that the majority is stablest theorem, which is a crucial ingredient in hardness of approximation results, is also true for our randomness efficient noise operator up to an additive error of  $2\epsilon$ . We state the original majority is stablest theorem below:

► **Theorem 4** (Majority Is Stablest [11]). *Let  $\mathbf{f} : \{-1, 1\}^n \rightarrow [0, 1]$  be a function with  $E[\mathbf{f}] = \mu$ . Suppose  $\text{Inf}_i^{\leq 10 \log(1/\tau)}(f) \leq \tau$  for all  $i \in [n]$ . Then:*

$$\langle \mathbf{f}, T_p \mathbf{f} \rangle \leq \Gamma_{1-2p}(\mu) + \frac{10 \log \log(1/\tau)}{(2p) \log(1/\tau)}$$

where  $\Gamma_{1-2p}$  is the Gaussian noise stability curve.

Our result shows that the stability of a function under our randomness efficient noise operator,  $\langle \mathbf{f}, T_{p,\epsilon}^{\text{sparse}} \mathbf{f} \rangle$  also obeys the same upper bound, with an extra additive error of  $2\epsilon$ .

As a secondary application, our construction also implies an explicitly constructible small set expander with large eigenvalues. We say that a graph  $G = (V, E)$  is a small set expander if for sufficiently small constant  $\delta$  and all subsets of vertices of size  $\delta|V|$ , the probability of leaving the set in one step of a random walk is at least some constant (say .9). Finding an efficient algorithm for deciding whether a graph is a small set expander remains an open problem. Arora, Barak, and Steurer [2] observed that there is an algorithm that can solve the small set expansion problem in time exponential in the number of eigenvectors of  $G$  that have eigenvalue greater than  $1 - \xi$ . Thus a natural question is how many such eigenvectors could a small set expander have? The noisy hypercube is one of the few “counterexamples” to the efficiency of the above mentioned algorithm, as it is an  $N$ -vertex graph that can have  $\text{polylog}(N)$  such eigenvectors. Our construction implies the existence of a “sparse” noisy hypercube with similar spectrum and small set expansion properties.

► **Theorem 5.** *For every  $\xi > 0$ , there is an explicit  $N$ -vertex small set expander with  $\text{polylog}(N)$  eigenvectors with eigenvalue  $1 - \xi$ . Moreover the graph contains*

$$O\left(N \log N \cdot \text{poly}\left(\left(\frac{1}{\xi} \log \log N\right)^{\log \frac{1}{\xi}}\right)\right)$$

edges.

The main interest in small set expansion is the relationship between the number of *vertices* and the number of large eigenvalues. Our construction does not improve on any lower bounds on the number of such eigenvalues a small set expander could have. However, we do note that our graph is sparse in the number of edges, containing about  $N \log N$  edges as opposed to the  $O(N^2)$  needed for the original noisy hypercube.

### 1.3 Background and Related Work

The idea of approximating nonuniform distributions such as  $\mathcal{D}_p$  is not entirely new in pseudorandomness. In fact, the linear tests on  $\mathcal{D}_p$  that we aim to fool are a special case of combinatorial shapes. An  $(m, n)$  combinatorial shape is a function  $f : [m]^n \rightarrow \{0, 1\}$  that can be expressed as  $f(x_1, \dots, x_n) = h(1_{A_1}, \dots, 1_{A_n})$  for some symmetric function  $h : \{0, 1\}^n \rightarrow \{0, 1\}$  and subsets  $A_1, \dots, A_n \subset [m]$ . By setting  $m = 1/p$  and  $h$  as the parity of all its inputs, we can express the parity of any  $I \subset [n]$  if we set  $A_i = \{1\}$  if  $i \in I$  and  $A_i = \emptyset$  otherwise. Gopalan, Meka, Reingold, and Zuckerman [6] give a PRG that fools all  $(m, n)$ -combinatorial shapes using seed length  $O(\log m + \log n + \log^2(1/\epsilon)) = O(\log 1/p + \log n + \log^2(1/\epsilon))$ . The main drawback of [6] that we improve on is that the seed length is only guaranteed to be  $O(\log n)$ , which implies only a polynomial sized construction. On the other hand, when  $p$  is a power of 2, our construction guarantees a nearly linear sized construction, with a slightly worse dependence on  $p$ , and a slightly better dependence on  $\epsilon$ .

In a previous work, Even, Goldreich, Luby, Nisan, and Veličković [5] study the approximation of distributions on  $[m]^n$  where each coordinate is an independent (and not necessarily identical) distribution. For any distribution  $\mathcal{D} = \mathcal{D}_1 \times \dots \times \mathcal{D}_n$  where each  $\mathcal{D}_i$  is independent, their constructions give sample spaces that have size  $(n/\epsilon)^{\log(1/\epsilon)}$  and  $(n/\epsilon)^{\log n}$  such that for any  $I \subset [n]$ , the marginal distribution of the sample space restricted to  $I$  is  $\epsilon$ -close to the marginal distribution of  $\mathcal{D}$  in max-norm.

Chin Ho Lee [9] gave a pseudorandom generator that fooled (under the **uniform** distribution) the XOR of any  $k$  boolean functions on disjoint inputs of length  $m$  with error  $\epsilon$  using seed length  $\tilde{O}(m + \log(km/\epsilon))$  where the  $\tilde{O}$  hides polynomial factors in  $\log m$ ,  $\log \log k$ , and

$\log \log(1/\epsilon)$ . By setting  $m = \log(1/p)$ ,  $k = n$ , and each of the  $k$  boolean functions as simply the product of its  $m$  bits, we get a pseudorandom generator that fools linear tests under the product distribution on  $n$  bits with marginals  $p$ . Indeed, for any linear test on  $n$  bits  $\sum_{i \in I} x_i$ ,  $I \subset [n]$ , we can consider the XOR of  $k = n$  boolean functions where each function  $f_i$  is the product of  $m$  bits if  $i \in I$  and is 0 otherwise. Then, Lee's generator produces a random variable  $X \sim \{0, 1\}^{km}$  that fools such a function under the uniform distribution. Thus the random variable  $Z \sim \{0, 1\}^n$  obtained by simply taking the product of the bits in each of the  $m$  disjoint blocks of  $X$  fools the original linear test with respect to the product distribution with marginals  $p$ . Lee's pseudorandom generator thus immediately gives a pseudorandom generator fooling linear tests with respect to the product distribution with marginals  $p$  with seed length  $\tilde{O}(\log(1/p) + \log(n/\epsilon))$ .

We mention that Meka, Reingold, and Tal [10] define a notion of " $\delta$ -biased distributions with marginals  $p$ ." However, their definition of approximation is ad hoc for their main goal of constructing PRGs for width-3 branching programs.

Our application of sparsifying the noisy hypercube is related to the classic result of Spielman and Teng in the edge sparsification of graphs [14]. Indeed, their sparsification algorithm, when run on the noisy hypercube, should produce a sparsified graph with the properties we aim to preserve. However, the main drawback to this approach is that the sparsification algorithm runs in time  $mpolylog(m)$  where  $m$  is the number of edges. In the case of the noisy hypercube, which is a dense graph defined on  $\{0, 1\}^n$ , this algorithm is much less efficient than the explicit construction we provide.

Barak et al. previously explored the idea of reducing the size of the noisy hypercube, which has close ties to hardness of approximation [3]. Their work presents a "derandomized noisy hypercube" along with the appropriate analogues of small set expansion and the majority is stablest theorem. As their interest was in the relationship between the number of *vertices* and the number of large eigenvalues of a small set expander, their constructed graph contains a reduced number of vertices. On the other hand, our construction keeps the same  $2^n$  vertices of the original noisy hypercube and reduces the number of edges.

## 1.4 Overview of Techniques

The construction of the randomness efficient noise operator and small set expanders are essentially direct applications of our construction of generalized small bias sets. Thus here we focus on the intuition behind our construction. It's easy to see that the bitwise product of  $\log_2(1/p)$  independent uniform samples from  $\{0, 1\}^n$  is exactly equivalent to  $\mathcal{D}_p$  for  $p$  a power of 2. Thus intuitively, if  $\epsilon$ -biased sets approximate the uniform distribution on  $\{0, 1\}^n$ , then the bitwise product of  $\log_2(1/p)$  random draws from an  $\epsilon$ -biased set should approximate  $\mathcal{D}_p$ . Our main construction formalizes this intuition by showing via a hybrid argument that such a bitwise product indeed fools linear tests when the input is drawn from  $\mathcal{D}_p$ . This simple idea is not sufficient however, as the final seed length will be roughly  $\log_2(1/p) \log n$  which implies at least a polynomial sized support for small  $p$ .

To improve the dependence on  $n$ , we observe that the parities of sufficiently large  $I \subset [n]$  will be close to uniform on  $\{0, 1\}$ . More specifically, the probability that the parity of a subset of indices  $I$  under the distribution  $\mathcal{D}_p$  is 1 is  $\frac{1}{2} - \frac{1}{2}(1 - 2p)^{|I|}$ . Thus for  $|I| \geq \frac{1}{2p} \ln(\frac{1}{\epsilon})$  the probability of the parity being 1 is  $\epsilon/2$  close to  $1/2$ . This means that we only need to accurately simulate the behavior of  $\mathcal{D}_p$  for  $|I|$  smaller than  $k = \frac{1}{2p} \ln(\frac{1}{\epsilon})$ . For large  $|I|$  we simply need to simulate the uniform distribution. To do so, we can take the bitwise AND of  $\log_2(1/p) - 1$  independent samples from a  $k$ -wise  $\epsilon$ -biased set (using seed length only  $\log \log n$ ). This simulates  $\mathcal{D}_{p/2}$ . Finally we take the bitwise product of this with a final

$\epsilon$ -biased set with seed length  $\log n$ . For small  $|I|$ , the behavior of the parities under  $\mathcal{D}_p$  are preserved, and for large  $|I|$ , the product of the  $k$ -wise  $\epsilon$ -biased sets will contain at least one 1, so the final probability the parity is 1 will be the probability that the final  $\epsilon$ -biased set outputs 1 on a specific coordinate, which is roughly  $1/2$ .

## 1.5 Paper Organization

In Section 2 we define the necessary preliminaries and notation. Section 3 presents and proves the correctness of our construction and Section 4 presents the applications of our result to randomness efficient noise and small set expansion. Finally in Section 5 we discuss lower bounds for our generalization of  $\epsilon$ -biased sets and further directions for research.

## 2 Preliminaries and Notation

In general we denote random variables as capital letters such as  $X$  and  $Y$ . We denote fixed values using lowercase such as  $x, y$ . Distributions are denoted with calligraphic capital letters such as  $\mathcal{D}$ , and the uniform distribution on a set  $S$  is denoted via  $\mathcal{U}(S)$ . We distinguish vector-valued random variables from scalars via boldface:  $\mathbf{X}, \mathbf{x}$ , and refer to a value at a specific index of a vector via the corresponding nonbolded symbol with subscript:  $X_i, x_i$ . Vectors in this paper generally take on values in the field  $\mathbb{F}_2$  and thus arithmetic is generally done modulo 2. We use  $\langle \cdot, \cdot \rangle$  to denote the inner product of two vectors modulo 2. Finally, we define the binary operation “ $\odot$ ” between two vectors as the entrywise product modulo 2. For example, for  $\mathbf{X} = (X_1, \dots, X_n)$  and  $\mathbf{Y} = (Y_1, \dots, Y_n)$ , we have:  $\mathbf{X} \odot \mathbf{Y} = (X_1 Y_1, \dots, X_n Y_n)$ . It is straightforward to verify that for any vectors  $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \{0, 1\}^n$ , we have:  $\langle \mathbf{x}, \mathbf{y} \odot \mathbf{z} \rangle = \langle \mathbf{x} \odot \mathbf{y}, \mathbf{z} \rangle$

We first define the bias of a subset according to a distribution.

► **Definition 6** (Bias). *Let  $I \subset [n]$  and  $\mathcal{D}$  be any distribution on  $\{0, 1\}^n$ . Then the bias of  $I$  according to  $\mathcal{D}$  is defined as*

$$b_{I, \mathcal{D}} = \Pr_{\mathbf{x} \sim \mathcal{D}} \left[ \bigoplus_{i \in I} x_i = 0 \right] - \Pr_{\mathbf{x} \sim \mathcal{D}} \left[ \bigoplus_{i \in I} x_i = 1 \right]$$

Equivalently, if  $\alpha \in \{0, 1\}^n$  then we say that the bias is:

$$b_{\alpha, \mathcal{D}} = \Pr_{\mathbf{x} \sim \mathcal{D}} [\langle \alpha, \mathbf{x} \rangle = 0] - \Pr_{\mathbf{x} \sim \mathcal{D}} [\langle \alpha, \mathbf{x} \rangle = 1]$$

When the probability distribution is clear from context, we denote the bias of  $I$  as  $b_I$ .

Next, we define the concept of  $\epsilon$ -biased sets and  $k$ -wise independent  $\epsilon$ -biased sets, both of which have already well known constructions, and are crucial for our construction of  $\epsilon$ -biased product distributions with marginals  $p$ .

► **Definition 7** ( $\epsilon$ -biased set). *An  $\epsilon$ -biased set is a small set  $S \subset \{0, 1\}^n$  such that for every  $\alpha \in \{0, 1\}^n$  we have:*

$$|b_{\alpha, \mathcal{U}(S)}| = \left| \Pr_{\mathbf{x} \sim \mathcal{U}(S)} [\langle \alpha, \mathbf{x} \rangle = 0] - \Pr_{\mathbf{x} \sim \mathcal{U}(S)} [\langle \alpha, \mathbf{x} \rangle = 1] \right| \leq \epsilon$$

or equivalently:

$$\left| \Pr_{\mathbf{x} \sim \mathcal{U}(S)} [\langle \alpha, \mathbf{x} \rangle = 1] - \Pr_{\mathbf{x} \sim \mathcal{U}(\{0, 1\}^n)} [\langle \alpha, \mathbf{x} \rangle = 1] \right| \leq \epsilon/2$$

Numerous works [12, 15] show that there are explicit constructions of  $\epsilon$ -biased sets that require  $\log n + O(\log \frac{1}{\epsilon})$  random bits to specify a random point in  $S$ , or in other words, the size of  $S$  is linear in  $n$ . A weaker notion of  $\epsilon$ -biased sets only considers the parity of subsets of indices of size at most  $k$ :

► **Definition 8** (*k*-wise  $\epsilon$ -biased set). A *k*-wise  $\epsilon$ -biased set is a small set  $S \subset \{0, 1\}^n$  such that for any  $\alpha \in \{0, 1\}^n$  with hamming weight  $|\alpha| \leq k$ . We have:

$$|b_{\alpha, U(S)}| = |Pr_{\mathbf{x} \sim U(S)}[\langle \alpha, \mathbf{x} \rangle = 0] - Pr_{\mathbf{x} \sim U(S)}[\langle \alpha, \mathbf{x} \rangle = 1]| \leq \epsilon$$

or equivalently:

$$|Pr_{\mathbf{x} \sim U(S)}[\langle \alpha, \mathbf{x} \rangle = 1] - Pr_{\mathbf{x} \sim U(\{0, 1\}^n)}[\langle \alpha, \mathbf{x} \rangle = 1]| \leq \epsilon/2$$

Naor and Naor show that there are explicit constructions of *k*-wise  $\epsilon$ -biased sets that require  $O(\log k + \log \log n + \log \frac{1}{\epsilon})$  random bits to specify a random point in  $S$ .

Our notion of approximating a product distribution with marginals  $p$  is the natural extension of the notion of approximation given by  $\epsilon$ -biased sets: the parity of any subset of coordinates from our approximate distribution should look like the parity of the subset of coordinates from  $\mathcal{D}_p$ .

► **Definition 9** ( $(p, \epsilon)$ -biased sample space). Let  $p \in [0, 1]$ . A  $(p, \epsilon)$ -biased sample space is a distribution  $\mathcal{Z}$  on  $\{0, 1\}^n$  with small support  $S \subset \{0, 1\}^n$  such that for every  $\alpha \in \{0, 1\}^n$  we have:

$$|Pr_{\mathbf{z} \sim \mathcal{Z}}[\langle \alpha, \mathbf{z} \rangle = 1] - Pr_{\mathbf{r} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{r} \rangle = 1]| \leq \epsilon$$

Historically, the definition of  $\epsilon$ -biased sets and *k*-wise independent  $\epsilon$ -biased sets use small bias as their notion of approximation. As stated in their definitions above, this notion is equivalent (up to constant factors) with the alternate notion that the distribution of the outputs of any linear function on input  $x \sim U(S)$  is close to the distribution when  $x \sim U(\{0, 1\}^n)$ . This equivalence no longer holds in the generalized notion of  $\epsilon$ -biased sets for  $\mathcal{D}_p$ . For example, if  $p$  is small, then the bias  $b_{I, \mathcal{D}_p}$  is almost 1 for any singleton subset  $I$ . The nonequivalence of these notions makes some simple facts about standard  $\epsilon$ -biased sets more tedious to prove for  $\epsilon$ -biased sets for  $\mathcal{D}_p$ . For completeness, we now state the facts important for our analysis, and defer their proofs to the appendix.

First, there is a well known relationship between the biases of a random  $x \in \{0, 1\}^n$  (over any distribution) and the probability mass function for the distribution.

► **Proposition 10.** Let  $\mathcal{D}$  be any distribution. For any  $\mathbf{a} \in \{0, 1\}^n$ , let  $p_{\mathbf{a}, \mathcal{D}}$  be the probability of sampling  $\mathbf{a}$  under  $\mathcal{D}$ . Let  $\mathbf{p}$  be the  $2^n$  length vector of probabilities  $p_{\mathbf{a}, \mathcal{D}}$  for each  $\mathbf{a}$ . Let  $\mathbf{b}$  be the  $2^n$  length vector of biases  $b_{\alpha, \mathcal{D}}$  indexed by  $\alpha \in \{0, 1\}^n$ . Let the Hadamard matrix  $H$  be the  $2^n \times 2^n$  matrix where each entry is defined as  $(-1)^{\langle \alpha, \mathbf{a} \rangle}$  then:

$$\mathbf{p} = 2^{-n} H^T \mathbf{b}$$

Given this proposition, we can prove a necessary fact for the analysis of our construction that if  $\mathcal{Z}$  is an  $(p, \epsilon)$ -biased space for  $\mathcal{D}_p$ , then  $\mathcal{Z}$  is close in max-norm to  $\mathcal{D}_p$ .

► **Corollary 11** ( $\epsilon$ -biased implies close in max norm). Let  $\mathcal{Z}$  be an  $(p, \epsilon)$ -biased sample space. Then  $\mathcal{Z}$  is  $2\epsilon$ -close to  $\mathcal{D}_p$  in max-norm. That is, for any  $\mathbf{a} \in \{0, 1\}^n$  we have:

$$|p_{\mathbf{a}, \mathcal{Z}} - p_{\mathbf{a}, \mathcal{D}_p}| \leq 2\epsilon$$

Finally, we note a useful fact that the distribution of the parity of *k* independent random variables in  $\{0, 1\}$  with marginals  $p$  is close to uniform on  $\{0, 1\}$  for sufficiently large *k*. The proof is again deferred to the appendix.

► **Proposition 12.** Consider *k* independent tosses of a biased coin with  $Pr[\text{Heads}] = p$ . Then the probability of an odd number of heads is  $\frac{1}{2} - \frac{1}{2}(1 - 2p)^k$ .

### 3 Construction

Our construction of a  $(p, \epsilon)$ -biased space for  $\mathcal{D}_p$  is as follows:

► **Construction 1.** Let  $k = \frac{1}{p} \ln \frac{100}{\epsilon}$  and  $t = \log_2 \frac{1}{2p}$ . Let  $\epsilon' = \frac{1}{100} \frac{2\epsilon}{t+1} = \frac{1}{100} \frac{2\epsilon}{\log_2 \frac{1}{p}} < \frac{\epsilon}{4} \leq \epsilon$ .

For  $1 \leq i \leq t$ , let  $\mathbf{X}_i$  be  $t$  independent draws from a  $k$ -wise  $\epsilon'$ -biased set of  $\{0, 1\}^n$ . We let  $\mathbf{X} = \bigodot_{i=1}^t \mathbf{X}_i$ . Let  $\mathbf{Y}$  be drawn from an  $\epsilon'$ -biased set of  $\{0, 1\}^n$ . Our final distribution is then  $\mathbf{Z} = \mathbf{X} \odot \mathbf{Y}$ .

We first state a main lemma that the product of  $\epsilon$ -biased spaces approximates  $\mathcal{D}_p$  with the right notion of approximation. We defer the proof to the appendix.

► **Lemma 13** (Coordinate-wise product of  $\epsilon$ -biased sets is  $\epsilon$ -biased for  $\mathcal{D}_p$ ). Let  $k \leq n$  and let  $\mathbf{X}_1, \dots, \mathbf{X}_t$  be independent draws from  $k$ -wise  $\epsilon$ -biased sets on  $\{0, 1\}^n$ . Then  $\mathbf{X} = \bigodot_i \mathbf{X}_i$  is a  $k$ -wise  $(\frac{1}{2^t}, t\epsilon/2)$ -biased sample space.

Given the lemma, we can then prove the correctness of our construction.

► **Theorem 14** (Main Result). Let  $0 < p < 1/2$ . For any  $\epsilon > 0$ ,  $\mathbf{Z}$  is a  $(p, \epsilon)$ -biased sample space requiring  $\log n + O(\log^2 \frac{1}{p} + \log \frac{1}{p} \log \frac{1}{\epsilon} + \log \frac{1}{p} \log \log n)$  uniform random bits to sample from.

**Proof.** We first note that using the constructions mentioned above, generating  $\mathbf{Z}$  requires  $\log n + O(t(\log k + \log \log n + \log \frac{1}{\epsilon'}) + \log \frac{1}{\epsilon'}) = \log n + O(\log^2 \frac{1}{p} + \log \frac{1}{p} \log \frac{1}{\epsilon} + \log \frac{1}{p} \log \log n)$  bits. Moreover, since the original constructions are explicit, we can construct the support of  $\mathbf{Z}$  via enumeration of all elements in each used  $\epsilon$ -biased set.

We claim that  $\mathbf{Z}$  is an  $\epsilon$ -biased distribution for  $\mathcal{D}_p$ . We show that for any  $\alpha \in \{0, 1\}^n$ :

$$|Pr_{\mathbf{z} \sim \mathbf{Z}}[\langle \alpha, \mathbf{z} \rangle = 1] - Pr_{\mathbf{r} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{r} \rangle = 1]| \leq \epsilon$$

The proof splits into two cases. For the first case, assume  $|\alpha| \leq k$ . Since  $\mathbf{Y}$  and the  $\mathbf{X}_i$ 's are  $k$ -wise  $\epsilon'$ -biased, by Lemma 13 we have immediately that:

$$|Pr_{\mathbf{z} \sim \mathbf{Z}}[\langle \alpha, \mathbf{z} \rangle = 1] - Pr_{\mathbf{r} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{r} \rangle = 1]| \leq (t+1) \frac{\epsilon'}{2} \leq \epsilon$$

In the second case, assume  $|\alpha| > k$ . Let  $I \subset [n]$  be any subset of the indices of size exactly  $k$  for which  $\alpha$  is 1. Consider the first component in the construction of  $\mathbf{Z}$ :

$$\mathbf{X} = \bigodot_{i=1}^t \mathbf{X}_i$$

where each  $\mathbf{X}_i \in \{0, 1\}^n$  is drawn from a  $k$ -wise  $\epsilon'$ -biased set. By Lemma 13, we know that the substring of  $\mathbf{X}$  restricted only to indices in  $I$ , denoted  $\mathbf{X}_I \in \{0, 1\}^k$ , is  $(\frac{p}{2}, \gamma)$ -biased for  $\mathcal{D}_{\frac{p}{2}, k}$  for  $\gamma \leq t\epsilon'/2 \leq \epsilon/100$ . Thus by Corollary 11, the distribution of  $\mathbf{X}_I$  is  $\frac{\epsilon}{50}$ -close to  $\mathcal{D}_{\frac{p}{2}, k}$  in max-norm. In particular, this means that:

$$Pr(\mathbf{X}_I = 0^k) \leq (1-p)^k + \frac{\epsilon}{50} \leq (1-p)^{\frac{1}{p} \ln \frac{100}{\epsilon}} + \frac{\epsilon}{50} = \frac{\epsilon}{100} + \frac{\epsilon}{50} \leq \frac{\epsilon}{4}$$

Thus with probability at least  $1 - \epsilon/4$ , the string  $\mathbf{X}$  will contain at least one 1 on an index where  $\alpha$  is 1. This means that we have:

$$\begin{aligned}
Pr_{\mathbf{z} \sim \mathbf{Z}}(\langle \alpha, \mathbf{z} \rangle = 1) &= P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X}_\alpha = 0^{|\alpha|}) + P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X}_\alpha \neq 0^{|\alpha|}) \\
&\leq \frac{\epsilon}{4} + \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X} = \mathbf{x}) \\
&= \frac{\epsilon}{4} + \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha, \mathbf{z} \rangle = 1 \mid \mathbf{X} = \mathbf{x})P(\mathbf{X} = \mathbf{x}) \\
&= \frac{\epsilon}{4} + \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha \odot \mathbf{x}, \mathbf{y} \rangle = 1)P(\mathbf{X} = \mathbf{x}) \\
&\leq \frac{\epsilon}{4} + \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} \left(\frac{1}{2} + \epsilon'\right) P(\mathbf{X} = \mathbf{x}) \\
&\leq \frac{1}{2} + \frac{\epsilon}{4} + \epsilon' \leq \frac{1}{2} + \frac{\epsilon}{2}
\end{aligned}$$

Similarly for a lower bound we have:

$$\begin{aligned}
Pr_{\mathbf{z} \sim \mathbf{Z}}(\langle \alpha, \mathbf{z} \rangle = 1) &= P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X}_\alpha = 0^{|\alpha|}) + P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X}_\alpha \neq 0^{|\alpha|}) \\
&\geq 0 + \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha, \mathbf{z} \rangle = 1 \wedge \mathbf{X} = \mathbf{x}) \\
&= \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha, \mathbf{z} \rangle = 1 \mid \mathbf{X} = \mathbf{x})P(\mathbf{X} = \mathbf{x}) \\
&= \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\langle \alpha \odot \mathbf{x}, \mathbf{y} \rangle = 1)P(\mathbf{X} = \mathbf{x}) \\
&\geq \left(\frac{1}{2} - \epsilon'\right) \sum_{\mathbf{x}: \mathbf{x}_\alpha \neq 0^{|\alpha|}} P(\mathbf{X} = \mathbf{x}) \\
&\geq \left(\frac{1}{2} - \epsilon'\right) \left(1 - \frac{\epsilon}{4}\right) \\
&= \frac{1}{2} + \epsilon' \frac{\epsilon}{4} - \epsilon' - \frac{\epsilon}{8} \\
&\geq \frac{1}{2} - \epsilon' - \frac{\epsilon}{4} \\
&\geq \frac{1}{2} - \frac{\epsilon}{2}
\end{aligned}$$

Combining the upper and lower bound shows that  $Pr_{\mathbf{z} \sim \mathbf{Z}}(\langle \alpha, \mathbf{z} \rangle = 1)$  is  $\epsilon/2$  close to  $1/2$ . Since by Proposition 12 we know that  $Pr_{\mathbf{r} \sim \mathcal{D}_p}(\langle \alpha, \mathbf{z} \rangle = 1)$  is also  $\epsilon/2$  close to  $1/2$  we must have that:

$$|Pr_{\mathbf{z} \sim \mathbf{Z}}(\langle \alpha, \mathbf{z} \rangle = 1) - Pr_{\mathbf{r} \sim \mathcal{D}_p}(\langle \alpha, \mathbf{r} \rangle = 1)| \leq \epsilon \quad \blacktriangleleft$$

## 4 Applications

We first define the noisy hypercube, which is a crucial graph in our applications and also an important graph in many areas of theoretical computer science.

► **Definition 15** (Noisy Hypercube Graph). *The  $p$ -noisy hypercube graph, which we denote  $T_p$ , is the graph on vertex set  $\{0, 1\}^n$  such that a random step from node  $\mathbf{a} \in \{0, 1\}^n$  is equivalent to picking  $\mathbf{r} \sim \mathcal{D}_p$  and moving to  $\mathbf{a} + \mathbf{r}$ .*

## 31:10 Randomness Efficient Noise Stability

Note that the transition matrix of  $T_p$  has no nonzero entries since there is a nonzero probability of reaching any node from any other node and is thus very dense. Our  $\epsilon$ -biased distribution for  $\mathcal{D}_p$  allows us to construct a sparse noisy hypercube that has similar properties to the original noisy hypercube but with fewer edges.

► **Definition 16** (Sparse Noisy Hypercube Graph). *Let  $\mathbf{Z}$  be an  $(p, \epsilon)$ -biased sample space. The sparse  $(p, \epsilon)$ -noisy hypercube graph, which we denote  $T_{p,\epsilon}^{sparse}$ , is the graph on vertex set  $\{0, 1\}^n$  such that a random step from node  $\mathbf{a} \in \{0, 1\}^n$  is equivalent to picking  $\mathbf{z} \sim \mathbf{Z}$  and moving to  $\mathbf{a} + \mathbf{z}$ .*

Because of the size of our construction's seed length, each row and column of the  $2^n \times 2^n$  transition matrix of  $T_{p,\epsilon}^{sparse}$  has  $\tilde{O}(n)$  nonzero entries when  $p$  and  $\epsilon$  are constant.

We first show that the noise operator defined by  $T_{p,\epsilon}^{sparse}$  has similar eigenvalues to that of the original noise operator. This leads to the fact that our randomness efficient notion of stability approximates the original notion of stability, and also implies that the graph  $T_{p,\epsilon}^{sparse}$  is our desired sparse small set expander.

### 4.1 Eigenvalues

The main feature about  $T_{p,\epsilon}^{sparse}$  from which our applications arise is that it has a similar spectrum to  $T_p$ . We first give a well known (and easily verifiable) fact about the eigenvalues and eigenvectors of graphs on the boolean hypercube that are defined like above.

► **Theorem 17.** *Let  $\mathbf{Z}$  be any distribution on  $\{0, 1\}^n$ . Define  $G = (V, E)$  on vertices  $V = \{0, 1\}^n$  as the graph on which a random step starting at  $\mathbf{a} \in \{0, 1\}^n$  is equivalent to drawing  $\mathbf{z} \in \mathbf{Z}$  and moving to  $\mathbf{a} + \mathbf{z}$ . Let  $M$  be the  $2^n \times 2^n$  transition matrix of  $G$ . For every subset of indices  $I \subset [n]$ , define the vector  $\mathbf{v}_I \in \{-1, 1\}^{2^n}$  to be 1 if the parity of the  $i$ th bitstring in  $\{0, 1\}^n$  restricted to  $I$  is 0 and  $-1$  if the parity is 1. Each  $\mathbf{v}_I$  is an eigenvector of  $M$  with eigenvalue  $b_{I,\mathbf{Z}}$ .*

Given this well known fact it is straightforward to see that the eigenvalue profiles of  $T_p$  and  $T_{p,\epsilon}^{sparse}$  are close:

► **Corollary 18.** *The graphs  $T_p$  and  $T_{p,\epsilon}^{sparse}$  have the same eigenvectors. For every eigenvector  $\mathbf{v}$  of both graphs, the corresponding eigenvalues differ by at most  $2\epsilon$ .*

**Proof.** By Theorem 17, both  $T_p$  and  $T_{p,\epsilon}^{sparse}$  have the same eigenvectors  $\mathbf{v}_I \in \{-1, 1\}^{2^n}$ . For any  $I$ ,  $\mathbf{v}_I$  has eigenvalue  $b_{I,\mathcal{D}_p}$  in  $T_p$  and  $b_{I,\mathbf{Z}}$  in  $T_{p,\epsilon}^{sparse}$  where  $\mathbf{Z}$  is an  $\epsilon$ -biased distribution for  $\mathcal{D}_p$ . However we know by definition of  $(p, \epsilon)$ -biased distribution that:

$$|b_{I,\mathbf{Z}} - b_{I,\mathcal{D}_p}| \leq 2\epsilon \quad \blacktriangleleft$$

### 4.2 Randomness Efficient Noise

The stability of a boolean function  $\mathbf{f}$  on  $\{-1, 1\}^n$  is a fundamental concept in the analysis of boolean functions that measures the tendency of the output of a function to change when each bit of the input is flipped independently with probability  $p$ . In our context, the stability is equivalent to

$$\text{Stab}_{1-2p} = \langle \mathbf{f}, T_p \mathbf{f} \rangle$$

where we think of  $\mathbf{f}$  as a  $2^n$  length truth table, and  $T_p$  is the transition matrix of the noisy hypercube above (here we no longer think of  $\langle \cdot, \cdot \rangle$  as the inner product modulo 2).

We can show that the stability of a function under our notion of “derandomized noise”, where noise is added to the input via a sample from a  $(p, \epsilon)$ -biased space for  $\mathcal{D}_p$  is close to the original notion of stability.

► **Theorem 19** (Randomness Efficient Noise Stability is Close to Noise Stability). *Let  $\mathbf{f} : \{-1, 1\}^n \rightarrow [0, 1]$  be a function with  $E[\mathbf{f}] = \mu$ . Then:*

$$\text{Stab}_{1-2p}(\mathbf{f}) - 2\epsilon \leq \text{Stab}_{1-2p}^{\text{sparse}}(\mathbf{f}) \leq \text{Stab}_{1-2p}(\mathbf{f}) + 2\epsilon$$

**Proof.** We can write  $\mathbf{f}$  in the Fourier basis as:

$$\mathbf{f} = \sum_I f_I \mathbf{v}_I$$

It is a well know fact in fourier analysis that:

$$\langle \mathbf{f}, T_p \mathbf{f} \rangle = \sum_I b_{I, \mathcal{D}_p} f_I^2$$

Similarly we can derive the corresponding expression for  $T_{p, \epsilon}^{\text{sparse}}$ :

$$\begin{aligned} & \langle \mathbf{f}, T_{p, \epsilon}^{\text{sparse}} \mathbf{f} \rangle \\ &= \left\langle \sum_I f_I \mathbf{v}_I, T_{p, \epsilon}^{\text{sparse}} \sum_I f_I \mathbf{v}_I \right\rangle \\ &= \left\langle \sum_I f_I \mathbf{v}_I, \sum_I b_{I, \mathbf{z}} f_I \mathbf{v}_I \right\rangle \\ &= \sum_I b_{I, \mathbf{z}} f_I^2 \langle \mathbf{v}_I, \mathbf{v}_I \rangle \\ &\leq \sum_I (b_{I, \mathcal{D}_p} + 2\epsilon) f_I^2 \\ &= \langle \mathbf{f}, T_p \mathbf{f} \rangle + 2\epsilon \end{aligned}$$

For the lower bound, we replace the inequality with  $b_{I, \mathbf{z}} \geq b_{I, \mathcal{D}_p} - 2\epsilon$  ◀

### 4.3 Small Set Expansion

We now show that our sparse noisy hypercube is our desired sparse small set expander with large eigenvalues. We first define the expansion of a graph.

► **Definition 20** (Expansion). *Given graph  $G = (V, E)$ , let  $S$  be any subset of vertices of  $G$ . The expansion of  $S$ , denote  $\Phi(S)$  is the probability that a randomly chosen edge  $(u, v)$  has  $v \notin S$  conditioned on  $u \in S$ . Equivalently, if  $G$  is a regular undirected graph, we have:*

$$\Phi_G(S) = \frac{E(S, V \setminus S)}{\sum_{v \in S} \text{deg}(v)}$$

In the context of small set expansion, we are typically interested in the expansion of sets that contain a small constant fraction  $\delta$  of vertices. We say that a graph is a small set expander if for sufficiently small  $\delta$ , all subsets containing  $\delta$ -fraction of vertices have expansion

## 31:12 Randomness Efficient Noise Stability

at least some constant (such as 0.9). We know that the noisy hypercube has  $n$  eigenvalues that are at least  $1 - 2p$ . As a consequence of Corollary 18, we know that  $T_{p,\epsilon}^{sparse}$  has at least  $n$  eigenvalues that are at least  $1 - 2p - 2\epsilon$ .

It remains to verify that the sparse noisy hypercube is also a small set expander. The following theorem relates the top eigenvectors of a graph to the expansion of sets [3].

► **Theorem 21.** *For any vector space  $\mathcal{V}$ , define the  $p \rightarrow q$  norm of a subspace  $\mathcal{U}$  of  $\mathcal{V}$  as:*

$$\|\mathcal{U}\|_{p \rightarrow q} = \max_{v \in \mathcal{V}} \frac{\|P_{\mathcal{U}}v\|_q}{\|v\|_p}$$

Where  $P_{\mathcal{U}}$  is the projection operator onto subspace  $\mathcal{U}$ .

For graph  $G = (V, E)$ , let  $\mathcal{U}$  be the subspace spanned by all eigenvectors of  $G$  with eigenvalue larger than  $\lambda$ . Then for any  $S \subset V$  containing  $\delta$  fraction of vertices we have:

$$\Phi(S) \geq 1 - \lambda - \|\mathcal{U}\|_{2 \rightarrow 4}^2 \sqrt{\delta}$$

In the case of the noisy hypercube, one can show via the Bonami Lemma that  $\|\mathcal{U}\|_{2 \rightarrow 4}$  is bounded. This implies via Theorem 21 that for sufficiently small  $\delta$ , the expansion of  $S$  is large. Finally, the next corollary relates the expansion of sets in  $T_{p,\epsilon}^{sparse}$  to those in  $T_p$ .

► **Corollary 22.** *Let  $\mathcal{U}_{true}$  be the subspace spanned by all eigenvectors of  $T_p$  with eigenvalue larger than  $\lambda$ . Let  $\mathcal{U}_{pseudo}$  be the subspace spanned by all eigenvectors of  $T_{p,\epsilon}^{sparse}$  with eigenvalue larger than  $\lambda + 2\epsilon$ . Then for any  $S \subset V$  that contains  $\delta$  fraction of vertices we have:*

$$\Phi_{T_{p,\epsilon}^{sparse}}(S) \geq 1 - \lambda - \|\mathcal{U}_{true}\|_{2 \rightarrow 4}^2 \sqrt{\delta} - 2\epsilon$$

**Proof.** Observe that since the eigenvalues of  $T_p$  are at most  $2\epsilon$  away from the eigenvalues of  $T_{p,\epsilon}^{sparse}$ , we have  $\mathcal{U}_{pseudo} \subset \mathcal{U}_{true}$ . This implies that  $\|\mathcal{U}_{pseudo}\|_{2 \rightarrow 4} \leq \|\mathcal{U}_{true}\|_{2 \rightarrow 4}$ . Thus by Theorem 21 we have:

$$\Phi_{T_{p,\epsilon}^{sparse}}(S) \geq 1 - (\lambda + 2\epsilon) - \|\mathcal{U}_{pseudo}\|_{2 \rightarrow 4}^2 \sqrt{\delta} \geq 1 - \lambda - \|\mathcal{U}_{true}\|_{2 \rightarrow 4}^2 \sqrt{\delta} - 2\epsilon \quad \blacktriangleleft$$

Thus sets in  $T_{p,\epsilon}^{sparse}$  have similar expansion to those in  $T_p$ . As mentioned earlier, by the Bonami Lemma [13], we have that when  $\lambda = (1 - 2p)^k$  then  $\|\mathcal{U}_{true}\|_{2 \rightarrow 4}^2 \leq 3^k$ . Thus we have:

$$\Phi_{T_{p,\epsilon}^{sparse}}(S) \geq 1 - (1 - 2p)^k - 3^k \sqrt{\delta} - 2\epsilon$$

Thus if we want expansion at least  $1 - \gamma$  for some small  $\gamma$ , we can set  $\epsilon < \frac{\gamma}{6}$ ,  $k > O\left(\frac{\ln 1/\gamma}{p}\right)$ , and  $\delta < \gamma^{O(\frac{1}{p})}$ .

## 5 Lower Bounds and Discussion

A natural question is how the size of our construction compares to an optimal, possibly nonexplicit construction. We first note that a simple probabilistic argument shows that any collection of  $2^n$  tests from  $\{0, 1\}^n$  to  $\{0, 1\}$  under the uniform distribution can be  $\epsilon$ -fooled by some function  $G : \{0, 1\}^s \rightarrow \{0, 1\}^n$  for  $s = \log n + 2 \log(1/\epsilon) + O(1)$ . The probabilistic construction is to simply pick each output of  $G$  independently and uniformly at random from  $\{0, 1\}^n$ . Using an analogous argument, picking each output of  $G$  independently from  $\mathcal{D}_p$  shows that there is a distribution  $\mathcal{Z}$  using the same seed length  $s$  that fools all  $2^n$  linear tests under  $\mathcal{D}_p$ . Thus, non-explicitly there exists a construction of an  $(p, \epsilon)$ -biased distribution whose size does not depend on  $p$ . Moreover, the distribution is *uniform* on its support, which is not the case for our explicit construction.

Alon et al [1] prove a lower bound of  $\Omega\left(\frac{n}{\epsilon^2 \log 1/\epsilon}\right)$  on the size of  $\epsilon$ -biased sets. We note that as a whole, since our construction works for  $p = 1/2$  this lower bound is also a lower bound in general for  $\epsilon$ -biased sets for  $\mathcal{D}_p$ . However, the story changes dramatically for small  $p$ . The previously mentioned lower bound is a result on the equivalence of  $\epsilon$ -biased sets with  $\epsilon$ -balanced linear error correcting codes. In an  $\epsilon$ -balanced linear error correcting codes with message length  $n$  and block length  $m$ , every codeword has weight between  $(1/2 - \epsilon)m$  and  $(1/2 + \epsilon)m$ . The equivalence between such codes and  $\epsilon$ -biased sets breaks down when generalizing to  $(p, \epsilon)$ -biased sample spaces. Under the assumption that we wish to construct an  $(p, \epsilon)$ -biased distribution for  $\mathcal{D}_{p,n}$  of size  $m$  that is uniform on its support, we would require a linear error correcting code with basis  $\mathbf{a}_1, \dots, \mathbf{a}_n \in \{0, 1\}^m$  such that the weight of the codeword  $\sum_I \mathbf{a}_I$  for every  $I \subset [n]$  is between  $\frac{1}{2} - \frac{1}{2}(1 - 2p)^{|I|} - \epsilon$  and  $\frac{1}{2} - \frac{1}{2}(1 - 2p)^{|I|} + \epsilon$ .

We note that our construction worsens in comparison to the optimal as  $p$  gets small. Indeed, as  $p$  approaches  $1/n$ , the amount of entropy in  $\mathcal{D}_p$  approaches 1, however, our seed length approaches  $\log^2 n$ . Thus, our construction illuminates a peculiar question about simulating an unfair coin: in order to simulate a coin with bias  $p$ , we require  $\log \frac{1}{p}$  flips of a fair coin, or in other words  $\log \frac{1}{p}$  bits of Shannon entropy. This is an extremely wasteful amount of randomness needed to simulate a distribution that has only  $H(p) \ll 1$  bits of entropy. However, it is unclear how to simulate an unfair coin using fair coins in a more efficient way. We note that the reverse direction of simulating a fair coin with a biased coin is a well known riddle attributed to von Neumann [16].

One reason that the efficiency of our construction depends on  $p$  is because of an asymmetry between the nature of the seed and the output. We aimed to use  $O(\log n)$  independent *fair* coin flips to approximate the distribution of  $n$  independent *unfair* coin flips. A more apt comparison would be to stretch  $O(\log n)$  unfair coins to approximate  $n$  unfair coins. It would be interesting to see whether there are simple constructions that can do so.

---

## References

- 1 Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost  $k$ -wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.
- 2 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015.
- 3 Boaz Barak, Parikshit Gopalan, Johan Håstad, Raghu Meka, Prasad Raghavendra, and David Steurer. Making the long code shorter, with applications to the unique games conjecture. *CoRR*, abs/1111.0405, 2011. URL: <http://arxiv.org/abs/1111.0405>.
- 4 Ben-Sasson, Sudan, Vadhan, and Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2003.
- 5 Even, Goldreich, Luby, Nisan, and Velickovic. Approximations of general independent distributions. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1992.
- 6 Parikshit Gopalan, Raghu Meka, Omer Reingold, and David Zuckerman. Pseudorandom generators for combinatorial shapes. *SIAM J. Comput.*, 42(3):1051–1076, 2013.
- 7 Johan Håstad. Testing of the long code and hardness for clique. In *Proceedings of The Twenty-Eighth Annual ACM Symposium On The Theory Of Computing (STOC '96)*, pages 11–19, New York, USA, May 1996. ACM Press.
- 8 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O’Donnell. Optimal inapproximability results for max-cut and other 2-variable CSPs? *SIAM J. Comput.*, 37(1):319–357, 2007.
- 9 Chin Ho Lee. Fourier bounds and pseudorandom generators for product tests. *arXiv preprint arXiv:1902.02428*, 2019.

- 10 Raghu Meka, Omer Reingold, and Avishay Tal. Pseudorandom generators for width-3 branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:112, 2018.
- 11 Elchanan Mossel, Ryan O’Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. *CoRR*, abs/math/0503503, 2005. URL: <http://arxiv.org/abs/math/0503503>.
- 12 Naor and Naor. Small-bias probability spaces: Efficient constructions and applications. *SICOMP: SIAM Journal on Computing*, 22, 1993.
- 13 Ryan O’Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. URL: <http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/analysis-boolean-functions>.
- 14 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.
- 15 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:41, 2017.
- 16 J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann’s Collected Works*, volume 5, pages 768–770. Pergamon, 1963.

## A Omitted Proofs

**Proof of Proposition 10.** We note that  $H^{-1} = 2^{-n}H^T$  and show that  $H\mathbf{p} = \mathbf{b}$ . For any fixed entry of  $\mathbf{b}$ , we have:

$$\begin{aligned}
 b_{\alpha, \mathcal{D}} &= Pr_{\mathbf{a} \sim \mathcal{D}} [\langle \alpha, \mathbf{a} \rangle = 0] - Pr_{\mathbf{a} \sim \mathcal{D}} [\langle \alpha, \mathbf{a} \rangle = 1] \\
 &= \sum_{\mathbf{a}: \langle \alpha, \mathbf{a} \rangle = 0} p_{\mathbf{a}, \mathcal{D}} - \sum_{\mathbf{a}: \langle \alpha, \mathbf{a} \rangle = 1} p_{\mathbf{a}, \mathcal{D}} \\
 &= \sum_{\mathbf{a}} (-1)^{\langle \alpha, \mathbf{a} \rangle} p_{\mathbf{a}, \mathcal{D}} \\
 &= (H\mathbf{p})_{\alpha}
 \end{aligned}$$

**Proof of Corollary 11.** For any  $\alpha \in \{0, 1\}^n$ , we have that:

$$\begin{aligned}
 &|b_{\alpha, \mathcal{Z}} - b_{\alpha, \mathcal{D}_p}| \\
 &= |Pr_{\mathbf{z} \sim \mathcal{Z}}[\langle \alpha, \mathbf{z} \rangle = 0] - Pr_{\mathbf{z} \sim \mathcal{Z}}[\langle \alpha, \mathbf{z} \rangle = 1] - (Pr_{\mathbf{z} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{z} \rangle = 0] - Pr_{\mathbf{z} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{z} \rangle = 1])| \\
 &\leq |Pr_{\mathbf{z} \sim \mathcal{Z}}[\langle \alpha, \mathbf{z} \rangle = 0] - Pr_{\mathbf{z} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{z} \rangle = 0]| + |Pr_{\mathbf{z} \sim \mathcal{Z}}[\langle \alpha, \mathbf{z} \rangle = 1] - Pr_{\mathbf{z} \sim \mathcal{D}_p}[\langle \alpha, \mathbf{z} \rangle = 1]| \\
 &\leq 2\epsilon.
 \end{aligned}$$

Fix any  $\mathbf{a} \in \{0, 1\}^n$ . By the formula from Proposition 10, we know that:

$$p_{\mathbf{a}, \mathcal{Z}} = 2^{-n} \sum_{\alpha \in \{0, 1\}^n} (-1)^{\langle \mathbf{a}, \alpha \rangle} b_{\alpha, \mathcal{Z}}$$

Similarly:

$$p_{\mathbf{a}, \mathcal{D}} = 2^{-n} \sum_{\alpha \in \{0, 1\}^n} (-1)^{\langle \mathbf{a}, \alpha \rangle} b_{\alpha, \mathcal{D}}$$

Thus:

$$\begin{aligned}
& |p_{\mathbf{a}, \mathcal{Z}} - p_{\mathbf{a}, \mathcal{D}}| \\
&= 2^{-n} \left| \sum_{\alpha \in \{0,1\}^n} (-1)^{\langle \mathbf{a}, \alpha \rangle} (b_{\alpha, \mathcal{Z}} - b_{\alpha, \mathcal{D}}) \right| \\
&\leq 2^{-n} \sum_{\alpha \in \{0,1\}^n} |(b_{\alpha, \mathcal{Z}} - b_{\alpha, \mathcal{D}})| \\
&\leq 2\epsilon
\end{aligned}$$

**Proof of Proposition 12.** Let  $X_i$  be a random variable with value  $-1$  if the  $i$ th coin toss is heads and  $1$  otherwise. Then the probability of an odd number of heads is equal to  $Pr[\prod_{i=1}^k X_i = -1]$ . Note that the random variable  $\frac{1}{2} + \frac{1}{2} \prod_{i=1}^k X_i$  is an indicator random variable that is  $1$  when there is an even number of heads. Thus

$$Pr(\text{even number of heads}) = \mathbb{E} \left[ \frac{1}{2} + \frac{1}{2} \prod_{i=1}^k X_i \right] = \frac{1}{2} + \frac{1}{2} \prod_{i=1}^k \mathbb{E}[X_i] = \frac{1}{2} + \frac{1}{2}(1 - 2p)^k$$

Thus the probability of an odd number of heads is

$$\frac{1}{2} - \frac{1}{2}(1 - 2p)^k$$

**Proof of Lemma 13.** We wish to show that for any  $\alpha \in \{0, 1\}^n$  with  $|\alpha| \leq k$ :

$$\left| Pr_{\mathbf{x} \sim \mathbf{X}}[\langle \alpha, \mathbf{x} \rangle = 1] - Pr_{\mathbf{x} \sim \mathcal{D}_{\frac{1}{2}^t}}[\langle \alpha, \mathbf{x} \rangle = 1] \right| \leq t\epsilon$$

We prove this via a hybrid argument.

Consider random variables  $\mathbf{X}_1, \dots, \mathbf{X}_t, \mathbf{R}_1, \dots, \mathbf{R}_t$  taking on values in  $\{0, 1\}^n$  where the  $X_i$ 's are independent draws from a  $k$ -wise  $\epsilon$ -biased set,  $R_i$ 's are chosen independently and uniformly at random from  $\{0, 1\}^n$ . We then define for  $0 \leq \ell \leq t$  the  $t+1$  hybrid distributions:

$$\mathcal{H}_\ell = \left\langle \alpha, \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+1}^t \mathbf{X}_i \right) \right\rangle$$

Notice that  $\mathcal{H}_0 = \langle \alpha, \mathbf{x} \rangle$  when  $\mathbf{x} \sim \mathbf{X}$ , while  $\mathcal{H}_{t+1} = \langle \alpha, \mathbf{x} \rangle$ , when  $x \sim \mathcal{D}_p$ . We show that  $|\mathcal{H}_\ell - \mathcal{H}_{\ell+1}| \leq \epsilon$  for every  $0 \leq \ell \leq t$ . Since each  $\mathcal{H}_\ell$  is a distribution on  $\{0, 1\}$  we can write the probability that distribution  $\mathcal{H}_\ell$  outputs  $1$  as:

31:16 Randomness Efficient Noise Stability

$$\begin{aligned}
& Pr_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+1}, \dots, \mathbf{X}_t}} \left[ \left\langle \alpha, \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+1}^t \mathbf{X}_i \right) \right\rangle = 1 \right] \\
&= Pr_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+1}, \dots, \mathbf{X}_t}} \left[ \left\langle \alpha, \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \mathbf{X}_{\ell+1} \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right) \right\rangle = 1 \right] \\
&= Pr_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+1}, \dots, \mathbf{X}_t}} \left[ \left\langle \alpha, \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right) \odot \mathbf{X}_{\ell+1} \right\rangle = 1 \right] \\
&= Pr_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+1}, \dots, \mathbf{X}_t}} \left[ \left\langle \alpha \odot \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right), \mathbf{X}_{\ell+1} \right\rangle = 1 \right] \\
&= \mathbb{E}_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+2}, \dots, \mathbf{X}_t}} \left[ Pr_{\mathbf{X}_{\ell+1}} \left[ \left\langle \alpha \odot \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right), \mathbf{X}_{\ell+1} \right\rangle = 1 \right] \right]
\end{aligned}$$

Where the last equality makes use of the fact that all the  $\mathbf{X}_i$ 's and  $\mathbf{R}_i$ 's are independent from each other. Similarly, we can write the probability that  $H_{\ell+1}$  outputs 1 as:

$$\begin{aligned}
& Pr_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_{\ell+1}, \\ \mathbf{X}_{\ell+2}, \dots, \mathbf{X}_t}} \left[ \left\langle \alpha, \left( \bigodot_{i=1}^{\ell+1} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right) \right\rangle = 1 \right] \\
&= \mathbb{E}_{\substack{\mathbf{R}_1, \dots, \mathbf{R}_\ell, \\ \mathbf{X}_{\ell+2}, \dots, \mathbf{X}_t}} \left[ Pr_{\mathbf{R}_{\ell+1}} \left[ \left\langle \alpha \odot \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right), \mathbf{R}_{\ell+1} \right\rangle = 1 \right] \right]
\end{aligned}$$

For fixed  $\mathbf{R}_1, \dots, \mathbf{R}_\ell$  and  $\mathbf{X}_{\ell+2}, \dots, \mathbf{X}_t$ , we know that  $\beta = \alpha \odot \left( \bigodot_{i=1}^{\ell} \mathbf{R}_i \right) \odot \left( \bigodot_{i=\ell+2}^t \mathbf{X}_i \right)$  is a vector with at most  $k$  1's. Thus since  $\mathbf{X}_{\ell+1}$  is  $k$ -wise  $\epsilon$ -biased, we know that:

$$|Pr_{\mathbf{X}_{\ell+1}} [\langle \beta, \mathbf{R}_{\ell+1} \rangle = 1] - Pr_{\mathbf{X}_{\ell+1}} [\langle \beta, \mathbf{X}_{\ell+1} \rangle = 1]| \leq \epsilon/2$$

Since expectation is just a weighted average, and each  $\mathcal{H}_\ell$  is a distribution over  $\{0, 1\}$ , we can conclude that  $|\mathcal{H}_\ell - \mathcal{H}_{\ell+1}| \leq \epsilon/2$ . Combining all the hybrid steps via triangle inequality gives us that  $|\mathcal{H}_0 - \mathcal{H}_\ell| \leq t\epsilon/2$   $\blacktriangleleft$

# Connectivity Lower Bounds in Broadcast Congested Clique

Shreyas Pai 

The University of Iowa, Iowa City, IA, USA  
shreyas-pai@uiowa.edu

Sriram V. Pemmaraju

The University of Iowa, Iowa City, IA, USA  
sriram-pemmaraju@uiowa.edu

---

## Abstract

We prove three new lower bounds for graph connectivity in the 1-bit broadcast congested clique model, BCC(1). First, in the KT-0 version of BCC(1), in which nodes are aware of neighbors only through port numbers, we show an  $\Omega(\log n)$  round lower bound for CONNECTIVITY even for constant-error randomized Monte Carlo algorithms. The deterministic version of this result can be obtained via the well-known “edge-crossing” argument, but, the randomized version of this result requires establishing new combinatorial results regarding the indistinguishability graph induced by inputs. In our second result, we show that the  $\Omega(\log n)$  lower bound result extends to the KT-1 version of the BCC(1) model, in which nodes are aware of IDs of all neighbors, though our proof works only for deterministic algorithms. This result substantially improves upon the existing  $\Omega(\log^* n)$  deterministic lower bound (Jurdiński et al., SIROCCO 2018) for this problem. Since nodes know IDs of their neighbors in the KT-1 model, it is no longer possible to play “edge-crossing” tricks; instead we present a reduction from the 2-party communication complexity problem PARTITION in which Alice and Bob are given two set partitions on  $[n]$  and are required to determine if the join of these two set partitions equals the trivial one-part set partition. While our KT-1 CONNECTIVITY lower bound holds only for deterministic algorithms, in our third result we extend this  $\Omega(\log n)$  KT-1 lower bound to constant-error Monte Carlo algorithms for the closely related CONNECTEDCOMPONENTS problem. We use information-theoretic techniques to obtain this result. All our results hold for the seemingly easy special case of CONNECTIVITY in which an algorithm has to distinguish an instance with one cycle from an instance with multiple cycles. Our results showcase three rather different lower bound techniques and lay the groundwork for further improvements in lower bounds for CONNECTIVITY in the BCC(1) model.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms; Theory of computation → Communication complexity; Mathematics of computing → Information theory

**Keywords and phrases** Distributed Algorithms, Broadcast Congested Clique, Connectivity, Lower Bounds, Indistinguishability, Communication Complexity, Information Theory

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.32

**Related Version** A full version of this paper is available at: <https://arxiv.org/abs/1905.09016>. A short version of this paper has appeared as a brief announcement in PODC 2019.

## 1 Introduction

We are given an  $n$ -node, completely connected *communication network* in which each node can broadcast at most  $b$  bits in each round. These  $n$  nodes and a subset of the edges of the communication network form the *input graph*. The question we ask is this: how many rounds of communication does it take to determine if the input graph is connected? This is the well known CONNECTIVITY problem in the  $b$ -bit *Broadcast Congested Clique*, i.e., the BCC( $b$ ) model.



© Shreyas Pai and Sriram V. Pemmaraju;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 32; pp. 32:1–32:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A series of recent rapid improvements [15, 13, 20] have shown that `CONNECTIVITY` and in fact `MST`, can be solved in  $O(1)$  rounds w.h.p.<sup>1</sup> in the  $b$ -bit *Congested Clique* model,  $CC(b)$ , when  $b = \log n$ . The  $CC(b)$  model allows each node to send a possibly different  $b$ -bit message to each of the other  $n - 1$  nodes in the network, in each round. In contrast, the fastest known algorithm for `CONNECTIVITY` in the  $BCC(\log n)$  model, due to Jurdziński and Nowicki [19], is deterministic and it runs in  $O\left(\frac{\log n}{\log \log n}\right)$  rounds. This contrast between  $BCC(b)$  and  $CC(b)$  is not surprising, given how much larger the overall bandwidth in  $CC(b)$  is compared to  $BCC(b)$ . Becker et al. [4] show that the *pair-wise set disjointness* problem can be solved in  $O(1)$  rounds in  $CC(1)$ , but needs  $\Omega(n)$  rounds in  $BCC(1)$ . But, despite the fact that `CONNECTIVITY` is such a fundamental problem, prior to this paper, only an  $\Omega(\log^* n)$ -round lower bound for deterministic algorithms for `CONNECTIVITY` in the  $KT-1$   $BCC(1)$  model was known [18].

Lower bound arguments in “congested” distributed computing models typically use a “bottleneck” technique [5, 8, 9, 11, 12, 16]. At a high level, this technique consists of showing that there is a low bandwidth cut in the communication network across which a high volume of information has to flow in order to solve the given problem. The lower bound on information flow is usually obtained via 2-party communication complexity lower bounds [23]. Not surprisingly, the “bottleneck” technique does not work in the  $CC(b)$  model because any cut with  $\Theta(n)$  vertices in each part, has a high bandwidth of  $\Theta(n^2 \cdot b)$  bits. In fact, a result of Drucker et al. [11], showing that circuits can be simulated efficiently in the Congested Clique model, indicates that no technique we currently know of can prove non-trivial lower bounds in the  $CC(b)$  model. However, as further shown by [11], “bottlenecks” are possible for some problems in the weaker  $BCC(b)$  model. In this model, every cut has bandwidth  $O(n \cdot b)$  and for example Drucker et al. [11] provide a reduction showing that for the problem of detecting the presence of a  $K_4$  in the input graph there is a cut across which  $\Omega(n^2)$  information has to flow. This leads to an  $\Omega(n/b)$  lower bound for  $K_4$ -detection in the  $BCC(b)$ .

All known lower bounds [11, 16] in the  $BCC(\log n)$  model have this general structure and these techniques work for problems such as fixed subgraph detection, all pairs shortest paths, diameter computation, etc., that are relatively difficult, requiring polynomially many rounds to solve. For “simpler” problems such as `CONNECTIVITY` and `MST`, we need more fine-grained lower bound techniques that allow us to prove polylogarithmic lower bounds. Specifically, since `CONNECTIVITY` can be solved in  $BCC(b)$  for any  $b \geq 1$  in just  $O(\text{poly}(\log n))$  rounds, the best we can expect is to show the existence of a cut across which  $\Omega(n \cdot \text{poly}(\log n))$  volume of information needs to flow. In fact, the connected components of a subgraph can be represented in  $O(n \log n)$  bits and this is all that needs to be communicated across a cut to solve `CONNECTIVITY`. Thus the best lower bound we can expect for `CONNECTIVITY` via this technique is an  $\Omega(\log n/b)$ . However, even this was unknown prior to this paper and one contribution of this paper is an  $\Omega(\log n/b)$  lower bound for `CONNECTIVITY` using the “bottleneck” technique.

## 1.1 Our Contribution

We consider the `CONNECTIVITY` problem and the closely related `CONNECTEDCOMPONENTS` problem in the  $BCC(1)$  model. In the latter problem, each node needs to output the label of the connected component it belongs to. We work in the  $BCC(1)$  model because it allows

---

<sup>1</sup> We use “w.h.p.” as short for “with high probability” which refers to the probability that is at least  $1 - 1/n^c$  for  $c \geq 1$ .

us to isolate barriers due to different levels of initial local knowledge (e.g., knowing IDs of neighbors vs not knowing IDs). This is also without loss of generality because a  $t$ -round lower bound in  $\text{BCC}(1)$  immediately translates to a  $t/b$ -round lower bound in  $\text{BCC}(b)$ . We consider two natural versions of the  $\text{BCC}(1)$  model, that we call KT-0 and KT-1 (using notation from [2]). In the KT-0 (“Knowledge Till 0 hops”) version, nodes are unaware of IDs of other nodes in the network and the  $n - 1$  communication ports at each node are arbitrarily numbered 1 through  $n - 1$ . In the KT-1 (“Knowledge Till 1 hop”) version, nodes know all  $n$  IDs in the network and the  $n - 1$  communication ports at each node are respectively labeled with the IDs of the nodes at the other end of the port. Note that if the bandwidth  $b = \Omega(\log n)$ , then there is essentially no distinction between the KT-0 and KT-1 versions since each node in the KT-0 version can send its ID to neighbors in constant rounds and then nodes would have as much knowledge as they initially do in the KT-1 version. But the difference in initial knowledge plays a critical role when  $b = o(\log n)$  and in fact our best results in these two models use completely different techniques. We present three main lower bound results in this paper, derived using very different techniques.

- In the KT-0 version of  $\text{BCC}(1)$  we show an  $\Omega(\log n)$  round lower bound for  $\text{CONNECTIVITY}$  even for constant-error randomized Monte Carlo algorithms. In fact, the lower bound is shown for the seemingly simpler “one cycle vs two cycles” problem in which the input graph is either a single cycle or consists of two disjoint cycles and the algorithm has to distinguish between these two possibilities. We use a well-known *indistinguishability* argument involving “edge crossing” [22, 3, 27] for this result, but the main novelty here is how this argument deals with the possibility that the algorithm can err on a constant fraction of the input instances. In a standard edge crossing argument one shows that for a particular YES instance (i.e., a connected or “one-cycle” instance)  $G$ , many of the NO instances  $G(e, e')$  obtained by crossing pairs of edges  $e$  and  $e'$  in  $G$  cannot be distinguished even after some  $t$  rounds of a  $\text{BCC}(1)$  algorithm (see Definition 7 for the precise definition of a crossing). But for a randomized lower bound in  $\text{BCC}(1)$ , it is not enough to consider a single YES instance. Instead, we use the bipartite *indistinguishability graph* induced by all YES and NO instances and show that this satisfies a polygamous version of Hall’s Theorem (see Theorem 1). This allows us to show the existence of a large generalized matching in the indistinguishability graph, which in turn shows that every  $o(\log n)$  round constant-error Monte Carlo algorithm can be fooled into making more errors than it is allowed.
- We then show that the above lower bound result extends to the KT-1 version of the  $\text{BCC}(1)$  model, though our proof only works for deterministic algorithms. This result substantially improves the  $\Omega(\log^* n)$ -round lower bound for deterministic algorithms for  $\text{CONNECTIVITY}$  in the KT-1  $\text{BCC}(1)$  model [18]. In KT-1, because of knowledge of IDs of neighbors, it is no longer possible to perform “edge crossing” tricks. But we are able to successfully use the “bottleneck” technique and show that there is a cut for the  $\text{CONNECTIVITY}$  problem across which  $\Omega(n \log n)$  bits need to flow. We prove this result by presenting a reduction from the 2-party communication complexity problem  $\text{PARTITION}$  [14]. In the  $\text{PARTITION}$  problem, we have a ground set  $[n]$  and Alice and Bob respectively are given two set partitions  $P_A$  and  $P_B$  of  $[n]$ . The goal is to output 1 iff  $P_A \vee P_B = \mathbf{1}$  where  $P_A \vee P_B$  (read as “ $P_A$  join  $P_B$ ”) is the finest partition  $P$  such that both  $P_A$  and  $P_B$  are refinements of  $P$ <sup>2</sup> and  $\mathbf{1}$  is the trivial partition consisting of the single

<sup>2</sup> Given two set partitions  $P$  and  $P'$  of  $[n]$ ,  $P$  is said to be a *refinement* of  $P'$  if for every part  $S \in P$ , there is a part  $S' \in P'$  such that  $S \subseteq S'$ . For example the partition  $(1, 2)(3, 4)(5)$  is a refinement of

set  $[n]$ . For example, if  $P_A = (1, 2)(3, 4)(5)$ ,  $P_B = (1, 2, 4)(3)(5)$ , and  $P_C = (1, 2, 4)(3, 5)$  then  $P_A \vee P_B = (1, 2, 3, 4)(5)$  and  $P_A \vee P_C = (1, 2, 3, 4, 5)$ . We then use the fact that the deterministic communication complexity of PARTITION is  $\Omega(n \log n)$  to obtain our result. Again, this time using a linear-algebraic argument, we show our result for a seemingly simple special case of CONNECTIVITY: “one cycle vs multiple cycles.” As far as we know, randomized communication complexity of PARTITION is a long-standing unresolved problem. Showing a lower bound on the randomized communication complexity of PARTITION will immediately lead to a KT-1 lower bound for randomized CONNECTIVITY algorithms, via our reduction.

- Our final result arises from our attempt to obtain a KT-1 lower bound even for constant-error Monte Carlo algorithms. We consider a version of the PARTITION problem, called PARTITIONCOMP, in which Alice and Bob are required to output the join of their respective input partitions  $P_A$  and  $P_B$  instead of just determining if  $P_A \vee P_B = \mathbf{1}$ . We use an information-theoretic argument to show that the mutual information of any algorithm, even a constant-error Monte Carlo algorithm, that solves this version of PARTITION is  $\Omega(n \log n)$ . This leads to an  $\Omega(\log n)$ -round lower bound for CONNECTEDCOMPONENTS in the KT-1 version of BCC(1), even for constant-error randomized Monte Carlo algorithms.

We prove in this paper non-trivial lower bounds for CONNECTIVITY in the BCC(1) model. The fact that our lower bounds hold even in the KT-1 model implies that the difficulty of the problem does not arise just from lack of knowledge of IDs of other nodes. The fact that our lower bounds hold for extremely sparse (i.e., 2-regular) graphs, suggests that there might be room to get stronger lower bounds by considering dense input graphs. In fact, using a deterministic sketching technique [25, 24], it is possible to obtain a deterministic  $O(\log n)$ -round BCC(1) algorithm for CONNECTIVITY for graphs with arboricity bounded by a constant. This implies that our lower bounds are tight for uniformly sparse graphs.

## 1.2 The BCC( $b$ ) Model

A size- $n$  KT-0 instance of the BCC(1) model consists of  $n$  vertices, each with a unique  $O(\log n)$ -bit ID. Each vertex has  $n - 1$  communication ports labeled distinctly, 1 through  $n - 1$ , in an arbitrary manner. A key feature of the KT-0 instance is that port labels have nothing to do with IDs. Pairs of communication ports are connected by network edges such that the underlying communication network is a clique. The  $n$  vertices along with a subset of the edges form the *input graph*. Thus some edges are both network edges and input graph edges, whereas the remaining edges are just network edges. The initial knowledge of a vertex  $v$  consists of its ID, its port numbering, an identification of ports that correspond to input edges, and an arbitrarily long string  $r_v$  of random bits. In each round  $t$ , each vertex  $u$  receives messages via broadcast from the remaining  $n - 1$  vertices in the previous round, performs local computation, and broadcasts a message of length at most  $b$ -bits. This message is received at the beginning of round  $t + 1$  by the remaining  $n - 1$  vertices along each of their communication ports that connect to  $u$ . After  $t$  rounds, the at most  $t \cdot b$  bits that  $v$  sends and the at most  $(n - 1) \cdot t \cdot b$  bits that  $v$  receives, along with the ports that they are received from make up the transcript of  $v$  at round  $t$ . A size- $n$  KT-1 instance of the BCC( $b$ ) model differs from a KT-0 instance in one important way: each network edge  $e = \{u, v\}$  is connected to  $u$  at port number  $ID(v)$  and connected to  $v$  at port number  $ID(u)$ . Thus, in a KT-1 instance, IDs serve as port numbers and the initial knowledge of a vertex consists include all  $n$  vertex IDs.

---

(1, 2)(3, 4, 5).

Since the main focus of the paper is to derive lower bounds, we assume the *public coin model* in which all the random strings  $r_v$  are identical. Lower bounds proved in the public coin model hold in the *private coin* model as well, in which all the  $r_v$ 's are distinct. For a decision problem, such as CONNECTIVITY, when we run a  $BCC(b)$  algorithm  $\mathcal{A}$  on an input graph  $G$ , each vertex outputs either YES or NO and the output of the system is YES if all vertices output YES and is NO otherwise. For a deterministic algorithm  $\mathcal{A}$  for CONNECTIVITY the system must output YES if  $G$  is connected and NO if  $G$  is disconnected. If  $\mathcal{A}$  is an  $\epsilon$ -error randomized Monte Carlo algorithm, then in order to be correct, it must satisfy the following requirements: (i) if  $G$  is connected then the system outputs YES with probability  $> 1 - \epsilon$  and (ii) if  $G$  is disconnected then the system outputs NO with probability  $> 1 - \epsilon$ .

### 1.3 Related Work

CONGEST model [28] lower bounds via the “bottleneck technique” that rely on communication complexity lower bounds have been shown for MST and related connectivity problems in [9] and for minimum vertex cover, maximum independent set, optimal graph coloring, all pairs shortest paths, and subgraph detection in [5, 8, 12]. This approach has also been used to derive  $BCC(\log n)$  lower bounds in [11, 16]. Becker et al. [4] define a spectrum of congested clique models parameterized by a *range* parameter  $r$ , denoting the number of distinct messages a node can send in a round. Setting  $r = 1$  gives us the  $BCC(b)$  model and setting  $r = n$  gives us the  $CC(b)$  model. They show the *pair-wise set disjointness* problem is sensitive to the value of  $r$  in the sense that for every pair of ranges  $r' < r$ , the problem can be solved provably faster in the model with range  $r$  than it can in the model with range  $r'$ .

Distributed lower bounds via the “edge crossing” argument have a long history in distributed computing, see [21] for an example in the context of proving message complexity lower bounds. More recent examples [22, 3, 27] appear in the context of *proof-labeling schemes*. Informally speaking, a *proof-labeling scheme* consists of a *prover* who labels the vertices of the input configuration with labels and a *distributed verifier* who is required to verify a predicate (e.g., do the marked edges form an MST?) in one round, using the help of the prover’s labels. The *verification complexity* of a proof-labeling scheme is the size of the largest message sent by the verifier. Patt-Shamir and Perry [27] show an  $\Omega(\log n)$  lower bound on the verification complexity of MST in the broadcast congested clique model. An  $\Omega(\log n)$  lower bound in the KT-0 version of  $BCC(1)$  for *deterministic* CONNECTIVITY algorithms follows from this result. The high level idea is that if there were a faster  $BCC(1)$  CONNECTIVITY algorithm, the prover could use the transcript of the algorithm at each vertex  $v$  as the label at  $v$ . The verifier could then broadcast these transcripts and locally, at each vertex  $v$ , simulate the algorithm at  $v$ . Baruch et al. [3] show that if there is a deterministic proof-labeling scheme with verification complexity  $\kappa$ , then there is a randomized proof-labeling scheme with one-sided error having verification complexity  $O(\log \kappa)$ . Combining this with the fact that MST verification has a deterministic proof-labeling scheme with  $O(\log^2 n)$  verification complexity [22], leads to a randomized proof-labeling scheme with  $O(\log \log n)$  verification complexity for MST [3, 27]. This needs to be contrasted with the fact that we show an  $\Omega(\log n)$  lower bound for CONNECTIVITY in KT-0  $BCC(1)$  even for constant-error Monte Carlo algorithms.

There have been recent attempts to combine the edge crossing and bottleneck techniques to obtain lower bounds for triangle detection in the CONGEST model [1, 12]. In particular, [12] provide an  $\Omega(\log n)$  lower bound for deterministic algorithms solving triangle detection in the KT-1 CONGEST model with 1-bit bandwidth.

Finally, lower bounds for the CONNECTIVITY problem are also known in related models like streaming and MPC [26, 6, 29]. Ideas in these papers, based on the polynomial method and boolean function complexity do not seem to imply any non-trivial lower bounds in the BCC(1) model.

## 2 Technical Preliminaries

**Polygamous Hall's Theorem.** Let  $G = (L, R, E)$  be a bipartite graph. A  $k$ -matching is a subgraph consisting of a set of nodes  $A \subseteq L$  where each  $v \in A$  has edges to nodes in the set  $nbr(v)$  such that  $|nbr(v)| = k$  and  $nbr(u) \cap nbr(v) = \emptyset$  for  $u, v \in A$ ,  $u \neq v$ . The size of a  $k$ -matching is the number of connected components in the subgraph.

► **Theorem 1 (Polygamous Hall's Theorem).** *Let  $G = (L, R, E)$  be a bipartite graph. If for every  $S \subseteq L$  we have  $|N(S)| \geq k|S|$  then  $G$  has a  $k$ -matching of size  $|L|$ .*

**Proof.** Make  $k$  copies of each node in  $L$  while keeping  $R$  the same. Now for every  $S \subseteq L$  we have  $|N(S)| \geq |S|$  and by Hall's marriage theorem, we have a matching in the modified bipartite graph which is a  $k$ -matching of size  $|L|$  in the original graph. ◀

**Yao's Minimax Theorem.** The standard way to prove lower bounds on  $\epsilon$ -error randomized algorithms is by invoking Yao's Minimax Theorem [31]. Let  $RR_\epsilon(P)$  denote the minimum round complexity of any  $\epsilon$ -error randomized algorithm that solves  $P$ . Let  $DR_\epsilon^\mu(P)$  denote the *distributional round complexity* of  $P$ , which is the minimum deterministic round complexity of an algorithm whose input is drawn from the distribution  $\mu$  (known to the algorithm) and the algorithm is allowed to make error on at most  $\epsilon$  fraction of the input (weighted by  $\mu$ ).

► **Theorem 2 (Yao's Minimax Theorem).** *For any problem  $P$ ,  $RR_\epsilon(P) \geq \max_\mu \{DR_\epsilon^\mu(P)\}$*

Yao's Minimax Theorem reduces the problem of proving a randomized lower bound to the task of designing a "hard" distribution that produces high distributional complexity.

**Lower bound for Partition.** The total number of distinct partitions on a ground set of  $n$  elements is given by the  $n^{\text{th}}$  Bell number  $B_n$ . It is well known that  $B_n = 2^{\Theta(n \log n)}$ . This means that the number of different possible input pairs that Alice and Bob can receive in the PARTITION problem is  $B_n^2 = 2^{\Theta(n \log n)}$ . Define the matrix  $M^n$  such that  $M^n(i, j) = 1$  if  $P_i \vee P_j = 1$  and  $M^n(i, j) = 0$  otherwise. Note that  $M^n$  is a  $B_n \times B_n$  matrix. Theorem 3 shows that this matrix is non-singular.

► **Theorem 3 ([10, 30]).**  *$\text{rank}(M^n) = B_n$  where  $B_n$  is the  $n^{\text{th}}$  Bell number*

Therefore by Lemma 1.28 of [23] we get the following corollary.

► **Corollary 4.** *The deterministic 2-party communication complexity of PARTITION is  $\Omega(n \log n)$*

**Information Theory.** Let  $\mu$  be a distribution over a finite set  $\Omega$  and let  $X$  be a random variable with distribution  $\mu$ . The *entropy* of  $X$  is defined as  $H(X) = -\sum_{x \in \Omega} \mu(x) \log \mu(x)$  and the *conditional entropy* of  $X$  given  $Y$  is  $H(X|Y) = \sum_y \Pr[Y = y] H(X|Y = y)$  where  $H(X|Y = y)$  is the entropy of  $X$  conditioned on the event  $\{Y = y\}$ . The *joint entropy* of two random variables  $X$  and  $Y$ , denoted by  $H(X, Y)$ , is the entropy of their joint distribution.

The *mutual information* between random variables  $X$  and  $Y$  is  $I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$  and the *conditional mutual information* between  $X$  and  $Y$  given  $Z$  is  $I(X; Y|Z) = H(X|Z) - H(X|Y, Z)$ . See the first two chapters of [7] for an excellent introduction to the basics of information theory.

### 3 Lower Bounds in the KT-0 model

This section is devoted to proving the following theorem. As mentioned earlier, our lower bound applies to the simpler “one cycle vs two cycles” problem which we will call `TWO CYCLE`. In this problem, the input is promised to be either a single cycle or two disconnected cycles, each of length at least 3 and the goal is to distinguish between these two types of inputs.

► **Theorem 5.** *For a sufficiently small constant  $0 < \epsilon \leq 1/2$ , the  $\epsilon$ -error randomized round complexity of the `TWO CYCLE` problem in the `BCC(1)` `KT-0` model is bounded below by  $\Omega(\log n)$ .*

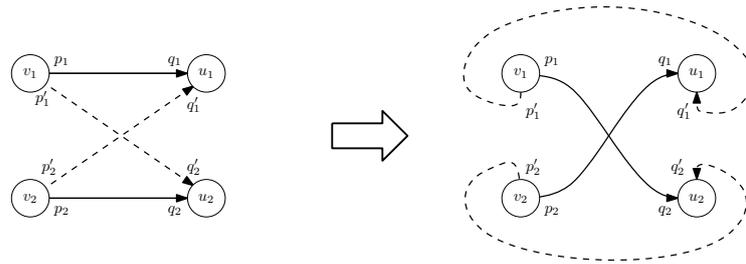
**Proof.** Consider an arbitrary one-cycle instance  $I_1 \in \mathcal{V}_1$  after  $t = 0.1 \log_3 n$  rounds of algorithm  $\mathcal{A}$ . Let  $x, y \in \{0, 1, \perp\}^t$  be the strings that correspond to the largest set of active edges after  $t$ -rounds of algorithm  $\mathcal{A}$ . We would like to count the size of this set of active edges. Recall that we orient each input graph edge of  $I_1$  in a clockwise direction. Therefore, each input graph edge in  $I_1$  can be labeled with a string of length  $2t$  which denotes messages sent across it from the head and the tail (in order) across the  $t$  rounds. This means that there are at least  $n/3^{2t} = n^{0.8}$  input graph edges in  $I_1$  that have the same messages sent across them. Therefore, the size of the set of active edges with respect to  $x, y$  is at least  $\Omega(n^{0.8})$ .

By Lemma 12 and Theorem 1, we can say that there exists a  $\Theta(\log n)$ -matching in  $\mathcal{G}_{x,y}^t$  of size  $|\mathcal{V}_1|$ . No matter what the algorithm  $\mathcal{A}$  outputs on any one-cycle instance, it will produce the same output on the matched  $O(\log n)$  two-cycle instances. By Lemma 13, we know that for any  $I_1 \in \mathcal{V}_1$  and  $I_2 \in \mathcal{V}_2$ ,  $\mu(I_1) = \mu(I_2) \cdot \Theta(\log n)$ . Therefore, each instance  $I_1 \in \mathcal{V}_1$  contributes to  $\Theta(\mu(I_1))$  the error of the algorithm which means that any  $t$ -round `BCC(1)` algorithm will have total error at least a constant. This implies the theorem. ◀

Two `KT-0` instances  $I_1$  and  $I_2$  are said to be *indistinguishable* after  $t$  rounds of an algorithm  $\mathcal{A}$  if the state of each vertex (i.e., the initial knowledge and the transcript at that vertex) after  $t$  rounds is the same in both the instances. We first introduce a technical tool called *indistinguishability via port-preserving crossings*. This tool has been used to show distributed computing lower bounds in several settings [21, 22, 3, 27] and we heavily borrow notation from [27]. For an edge  $e = (v, u)$  we use the notation  $e(p, q)$  to denote that  $e$  is connected to port  $p$  at  $v$  and to port  $q$  at  $u$ . For this notation to be unambiguous, we must think of the edge  $e = (v, u)$  as a directed edge  $v \rightarrow u$  even though the graph itself is undirected.

► **Definition 6** (Independent Edges [27]). *Let  $I$  be an instance with input graph  $G = (V, E)$  and let  $e_1 = (v_1, u_1)$  and  $e_2 = (v_2, u_2)$  be two edges of  $G$ . The edges  $e_1$  and  $e_2$  are said to be independent if and only if  $v_1, u_1, v_2, u_2$  are four distinct vertices and  $(v_1, u_2), (v_2, u_1) \notin E$ . A set of input graph edges is called independent if every pair of edges in the set is a pair of independent edges.*

► **Definition 7** (Port-Preserving Crossing [27]). *Consider an instance  $I$  with input graph  $G = (V, E)$ . Let  $e_1 = (v_1, u_1)$  and  $e_2 = (v_2, u_2)$  be two independent edges of  $G$ , and let  $e'_1 = (v_1, u_2)$  and  $e'_2 = (v_2, u_1)$  be two corresponding network edges in  $I$ . Let  $p_1, p_2, q_1, q_2, p'_1, q'_1, p'_2, q'_2$  be eight ports such that  $e_1(p_1, q_1), e_2(p_2, q_2), e'_1(p'_1, q'_2), e'_2(p'_2, q'_1)$ . The crossing of  $e_1$  and  $e_2$  in  $I$ , denoted by  $I(e_1, e_2)$ , is the instance obtained from  $I$  by replacing  $e_1$  and  $e_2$  in  $G$  with the edges  $e'_1$  and  $e'_2$  and rewiring the edges so that  $e_1(p'_1, q'_1), e_2(p'_2, q'_2), e'_1(p_1, q_2)$ , and  $e'_2(p_2, q_1)$ . (See Figure 1.)*



■ **Figure 1** This figure illustrates definition of a port-preserving crossing as per Definition 7.

The following lemma establishes a standard connection between indistinguishability and port-preserving crossings (henceforth “crossings”) and is in fact the main motivation for defining crossings. For simplicity, we say that a node sends the character  $\perp$  to denote the fact that the node remains silent. Therefore, the events of a node broadcasting a 0, a 1, or remaining silent can be described as sending the characters 0, 1, or  $\perp$  respectively.

► **Lemma 8.** *Let  $I$  be an instance with input graph  $G = (V, E)$  and let  $e_1 = (v_1, u_1)$  and  $e_2 = (v_2, u_2)$  be two independent edges of  $G$ . If  $v_1, v_2$  send the same sequence  $x \in \{0, 1, \perp\}^t$  and  $u_1, u_2$  send the same sequence  $y \in \{0, 1, \perp\}^t$  in the first  $t$  rounds of the algorithm, then  $I$  is indistinguishable from  $I(e_1, e_2)$  after  $t$  rounds.*

**Proof.** We will prove the lemma by induction on  $t$ . The initial knowledge of each vertex in  $I$  and  $I(e_1, e_2)$  is the same so the statement is true for  $t = 0$ .

Assume that the lemma is true for some round  $0 \leq i \leq t$ . Therefore, the characters broadcast by the vertices in round  $i + 1$  will be the same in both the instances. From the definition of port preserving crossing it is clear that  $I$  and  $I(e_1, e_2)$  differ only in four edges,  $e_1, e_2, e'_1 = (v_1, u_2)$ , and  $e'_2 = (v_2, u_1)$ . Therefore, all vertices except  $v_1, v_2, u_1$ , and  $u_2$  will receive the same characters across all their ports in round  $i + 1$  in both the instances and hence will have the same state in both instances after round  $i + 1$ .

Let the port names of the four edges in  $I$  and  $I(e_1, e_2)$  be as in Definition 7 and Figure 1. In  $I$ , the vertex  $u_1$  will receive the characters broadcast by  $v_1, v_2$  through ports  $q_1, q'_1$  respectively and in  $I(e_1, e_2)$  it will receive the characters broadcast by  $v_2, v_1$  through ports  $q_1, q'_1$  respectively. Note that  $v_1$  and  $v_2$  broadcast the same message in round  $i + 1$  since they send the same sequence  $x$  in the first  $t$  rounds and therefore, the state of  $u_1$  after round  $i + 1$  will be the same in both instances. We can make similar arguments for  $u_2, v_1$ , and  $v_2$  as well. Therefore, the state of each vertex after round  $i + 1$  is the same in both  $I$  and  $I(e_1, e_2)$  which proves the induction step as well as the lemma. ◀

As a “warm-up”, we first sketch an easy  $\Omega(\log n)$  lower bound for randomized Monte Carlo algorithms that make *polynomially small error*, i.e., error  $\epsilon = 1/n^c$  for constant  $c > 0$ . By Yao’s minimax theorem (Theorem 2), it suffices to show a lower bound on the distributional complexity of a deterministic algorithm under a hard distribution. Consider the following hard distribution  $\mu$ : Let  $I$  be an arbitrary instance such that the input graph  $G$  of  $I$  is a one-cycle on  $n$  vertices. Let  $S$  be an arbitrarily chosen set of exactly  $\lfloor n/3 \rfloor$  independent edges<sup>3</sup> and let  $I(S)$  be the set of all instances  $I(e, e')$  where  $e, e' \in S$ , and therefore,  $|I(S)| = \binom{\lfloor n/3 \rfloor}{2} = \Theta(n^2)$ . The hard distribution  $\mu$  places probability mass  $1/2$  on the

<sup>3</sup> Adding an edge to  $S$  invalidates at most two other edges, and therefore we can always find an independent set  $S$  of size  $\lfloor n/3 \rfloor$ .

instance  $I$  and uniformly distributes the remaining probability mass among the instances in  $I(S)$ . Now, given a  $t$ -round deterministic algorithm  $\mathcal{A}$  we can assign a  $2t$ -character label to each edge  $(v, u)$  obtained by concatenating the  $t$  characters broadcast by  $v$  and  $u$ . Here each character in the label belongs to the alphabet  $\{0, 1, \perp\}$ . The pigeon-hole principle implies that there is a set  $S' \subseteq S$ ,  $|S'| \geq n/(3 \cdot 3^{2t})$ , of edges in  $S$  with identical labels. Then by Lemma 8, for any  $e, e' \in S'$ ,  $I$  and  $I(e, e')$  are indistinguishable after  $t$ -rounds of  $\mathcal{A}$ . Since  $\mathcal{A}$  cannot make an error on  $I$ , it makes errors on all instances  $I(e, e')$  where  $e, e' \in S'$ . Since  $\mu$  assigned the probability mass  $1/2$  uniformly to all instances in  $I(S)$ , the probability that  $\mathcal{A}$  makes an error is at least  $|I(S')|/(2|I(S)|) = \binom{|S'|}{2}/\binom{\lfloor n/3 \rfloor}{2} \geq \Omega(3^{-4t})$ . Therefore, if  $t \leq 0.001 \cdot c \cdot \log_3 n$ , this error becomes  $\Omega(1/n^{0.001c})$  which is much larger than  $1/n^c$ ; a contradiction, implying that  $t > 0.001 \cdot c \cdot \log n$  and leading to the following theorem.

► **Theorem 9.** *For any constant  $c > 0$ , if  $\epsilon \leq 1/n^c$  then the  $\epsilon$ -error randomized round complexity of the CONNECTIVITY problem in the BCC(1) KT-0 model is  $\Omega(c \cdot \log n)$ .*

**Proof.** Note that since the probability mass on  $I$  is so large, any algorithm with permissible error probability must output YES on  $I$  and therefore, it will also output YES on all instances that are indistinguishable from  $I$ .

Given a  $t$ -round deterministic algorithm  $\mathcal{A}$  we can assign a  $2t$ -character label to each edge  $(v, u)$  where each character belongs to the alphabet  $\{0, 1, \perp\}$ . The label is assigned such that the head  $v$  sends the  $i^{\text{th}}$  character of the label and the tail  $u$  sends the  $(t+i)^{\text{th}}$  character of the label in round  $i$  for all edges. By using the pigeon hole principle, we see that there is a set  $S' \subseteq S$ ,  $|S'| \geq n/(3 \cdot 3^{2t})$ , of edges in  $S$  with identical labels. By Lemma 8, for any  $e, e' \in S'$ ,  $I$  and  $I(e, e')$  are indistinguishable after  $t$ -rounds of  $\mathcal{A}$ . Therefore, any  $t$  round algorithm will make an error on instances  $I(e, e')$  where  $e, e' \in S'$  and this makes the error at least  $\binom{|S'|}{2}/\binom{\lfloor n/3 \rfloor}{2} \geq \Omega(3^{-4t})$ . Therefore, if  $t \leq 0.001 \cdot c \cdot \log_3 n$ , this error becomes  $\Omega(1/n^{0.001c})$  which is much larger than  $1/n^c$ . ◀

The hard distribution  $\mu$  that led to the above theorem fails to give even a super-constant round lower bound for constant error probability. This is because for any constant  $\epsilon$ , there is a constant  $t$  such that the error probability  $|I(S')|/(2|I(S)|)$  of algorithm  $\mathcal{A}$  is smaller than  $\epsilon$ , leading to no contradiction.

### 3.1 A Lower Bound for Constant Error Probability

To get around this problem, we start with the observation that a two-cycle instance  $I(e, e')$  obtained from  $I$ , can also be obtained by crossing edges in other one-cycle instances, i.e.,  $I(e, e') = I'(f, f')$  for edges  $f, f'$  in an instance  $I' \neq I$ . Thus, as the algorithm executes, even though  $I(e, e')$  ceases to be indistinguishable from  $I$ , it may continue to be indistinguishable from  $I'$ . This suggests that we should be considering all one-cycle and two-cycle instances and all the edge crossings that lead from one-cycle instances to two-cycle instances. This motivates the definition below of a bipartite *indistinguishability graph* with all one-cycle and two-cycle instances as vertices. In the proof of Theorem 9, when we placed the entire probability mass on a single “star” indistinguishability graph with  $I$  being the central node and instances in  $I(S)$  being the leaves, we ran into trouble because the degree of  $I$  in this “star” shrank too quickly with the number of rounds,  $t$ . If we consider the full indistinguishability graph, we have more leeway. Specifically, showing the existence of a large matching in the indistinguishability graph would be helpful since the algorithm is forced to make an error at one of the two endpoints of each matching edge. We formalize this intuition below.

## 32:10 Connectivity Lower Bounds in Broadcast Congested Clique

Let the set of distinct one-cycle and two-cycle instances be  $\mathcal{V}_1$  and  $\mathcal{V}_2$  respectively let  $\mu$  be a probability distribution on these. Let  $\mathcal{A}$  be a  $t$ -round deterministic KT-0 algorithm which solves the TWOCYCLE problem correctly on  $(1 - \epsilon)$  fraction of input in the support of  $\mu$  (recall,  $\epsilon$  is a constant). For any instance  $I \in \mathcal{V}_1 \cup \mathcal{V}_2$ , call an edge  $e = (v, u)$  in the input graph of  $I$  *active* with respect to strings  $x, y \in \{0, 1, \perp\}^t$  iff  $v$  broadcasts the sequence given by  $x$  and  $u$  broadcasts the sequence given by  $y$  in the first  $t$  rounds of the algorithm  $\mathcal{A}$ . We call an edge active if the strings  $x, y$  are clear from the context.

► **Definition 10 (Indistinguishability Graph).** *Let  $t$  be a non-negative integer and let  $x, y \in \{0, 1, \perp\}^t$  be two strings of length  $t$ . The indistinguishability graph with respect to messages  $x$  and  $y$  after  $t$  rounds of algorithm  $\mathcal{A}$  is a bipartite graph  $\mathcal{G}_{x,y}^t = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}^t)$  where  $\mathcal{V}_1$  is the set of all one-cycle instances and  $\mathcal{V}_2$  is the set of all two-cycle instances and there is an edge  $\{I_1, I_2\} \in \mathcal{E}^t$  iff  $I_1 \in \mathcal{V}_1$  and  $I_2 \in \mathcal{V}_2$  and there exist two active independent directed edges  $e_1 = (v_1, u_1)$  and  $e_2 = (v_2, u_2)$  in the input graph of  $I_1$  such that  $I_2 = I_1(e_1, e_2)$ .*

We now propose to use a rather natural hard distribution  $\mu$  that assigns probability mass  $1/2$  distributed uniformly among the instances in  $\mathcal{V}_1$  and the remaining probability mass  $1/2$  distributed uniformly among the instances in  $\mathcal{V}_2$ . We first prove Lemma 11 that plays a crucial role in our overall proof by essentially showing that every one-cycle instance has sufficiently many two-cycle neighbors in  $\mathcal{G}_{x,y}^t$  with high degree. This in turn is used in Lemma 12 to prove that a Polygamous Hall's Theorem (Theorem 1) condition holds for  $\mathcal{G}_{x,y}^t$ . This allows us to show that  $\mathcal{G}_{x,y}^t$  can be packed with  $|\mathcal{V}_1|$  "stars," each with  $\Theta(\log n)$  leaves. We need this generalized notion of a matching because as shown in Lemma 13,  $|\mathcal{V}_2| = |\mathcal{V}_1| \cdot \Theta(\log n)$ . Therefore, the probability mass assigned to an instance in  $\mathcal{V}_2$  is  $1/\Theta(\log n)$  fraction of the probability mass assigned to an instance in  $\mathcal{V}_1$ . Thus, a "star" with its central node from  $\mathcal{V}_1$  and  $\Theta(\log n)$  leaves from  $\mathcal{V}_2$  has roughly equal probability mass assigned to the YES instance and NO instances.

► **Lemma 11.** *Consider an arbitrary instance  $I_1 \in \mathcal{V}_1$  that is a vertex of  $\mathcal{G}_{x,y}^t$ . If  $d \geq 1$  is the number of active edges of  $I_1$  with respect to  $x, y$  then for every  $i, 3 \leq i \leq d/2$ ,  $I_1$  has at least  $d/2$  neighbors of degree  $i \cdot (d - i)$ .*

**Proof.** A two-cycle instance  $I_2 \in \mathcal{V}_2$  will be a neighbor of  $I_1$  iff  $I_1$  and  $I_2$  form a pair of crossed instances with respect to  $x, y$ . Say  $I_2 = I_1(e, e')$  where  $e = (v, u)$  and  $e' = (v', u')$ . Note that  $I_2$  will have two new input graph edges  $(v, u')$  and  $(u, v')$  both of which are active and all input graph edges of  $I_1$  except for  $e, e'$  appear in the input graph of  $I_2$ . Therefore,  $I_2$  also has  $d$  active edges with respect to  $x, y$ . The degree of  $I_2$  is determined by the number of active edges either cycle, i.e., if  $I_2$  has  $i$  active edges in one cycle and  $d - i$  active edges in the other cycle then its degree in  $\mathcal{G}_{x,y}^t$  is  $i \cdot (d - i)$  since we can take one active edge from either cycle and cross them to produce a unique neighbor of  $I_2$ .

For every active edge  $e$  in the input graph of  $I_1$ , we can associate a unique active edge  $e_i$  such that  $I_1(e, e_i)$  has  $i$  active edges in one cycle and  $d - i$  active edges in the other cycle. Therefore,  $I_1$  has exactly  $d$  (or  $d/2$  if  $i = d/2$ ) neighbors having degree  $i(d - i)$ . This argument may not hold exactly for  $i = 1, 2$  because  $e$  and  $e_i$  as described need not form a pair of independent edges in this case. Thus, the lemma follows. ◀

► **Lemma 12.** *For the graph  $\mathcal{G}_{x,y}^t$ , consider an arbitrary set  $\mathcal{S} \subseteq \mathcal{V}_1$  of one-cycle instances with degree at least 1. Let  $N(\mathcal{S})$  be the neighborhood of  $\mathcal{S}$  in  $\mathcal{G}^t$ . Then  $|N(\mathcal{S})| \geq |\mathcal{S}| \cdot \Theta(\log d)$  where  $d$  is the smallest number of active edges in any instance in  $\mathcal{S}$ .*

**Proof.** Every  $I \in \mathcal{S}$  has at least  $d$  active edges, therefore by Lemma 11, there are at least  $d/2$  neighbors of  $I$  having degree  $i \cdot (d - i)$  for  $3 \leq i \leq d/2$ . Thus there are at least  $(d/2) \cdot |\mathcal{S}| / (i \cdot (d - i)) = \Theta(|\mathcal{S}|/i)$  two-cycle instances in  $N(\mathcal{S})$  having degree  $i \cdot (d - i)$ . Therefore, we have  $|N(\mathcal{S})| \geq \sum_{i=3}^{d/2} \Theta(|\mathcal{S}|/i) = |\mathcal{S}| \cdot \Theta(H_{d/2} - 3/2) \geq |\mathcal{S}| \cdot \Theta(\log d)$ , where  $H_n$  is the  $n^{\text{th}}$  harmonic number.  $\blacktriangleleft$

► **Lemma 13.**  $|\mathcal{V}_2| = |\mathcal{V}_1| \cdot \Theta(\log n)$ .

**Proof.** Let  $\mathcal{G} = \mathcal{G}_{\lambda, \lambda}^0$  ( $\lambda$  is the empty string) be the indistinguishability graph at round 0. Note that in  $\mathcal{G}$ , every instance in  $\mathcal{V}_1 \cup \mathcal{V}_2$  has strictly positive degree since each instance has  $n$  active edges. Therefore, we have  $|\mathcal{V}_1| = |N(\mathcal{V}_2)|$  and  $|\mathcal{V}_2| = |N(\mathcal{V}_1)|$ . Therefore, by Lemma 12, we have  $|\mathcal{V}_2| = |\mathcal{V}_1| \cdot \Omega(\log n)$ . Now we show that  $|\mathcal{V}_2| = |\mathcal{V}_1| \cdot O(\log n)$ .

Since each instance has  $n$  active edges, each one-cycle instance  $I_1$  has degree  $n(n - 3)/2$  because for each input graph edge  $e$  of  $I_1$  there are  $(n - 3)$  active edges independent of  $e$ , which we can cross with to get a unique neighbor of  $I_1$ . We need to divide by a factor of two because  $I_1(e, e') = I_1(e', e)$ . And each two-cycle instance  $I_2$  with the smaller cycle having length  $i$  has degree  $i \cdot (n - i)$  since we can cross any two edges in different cycles to get a neighbor of  $I_2$ .

Let  $\mathcal{T}_i$  denote the set of two-cycle instances with the smaller cycle having length  $i$  for  $3 \leq i \leq n/2$ .

For every input graph edge  $e$  in a one-cycle instance  $I$ , there is exactly one input graph edge  $e_i$  such that  $I(e, e_i) \in \mathcal{T}_i$ . Therefore, for  $3 \leq i < n/2$ , each one cycle instance has  $n$  neighbors such that the smaller cycle is of length  $i$ . And if  $n$  is even, each one-cycle instance will have  $n/2$  neighbors where both cycles have length  $n/2$  instead.

We will now show that  $|\mathcal{T}_i| \leq |\mathcal{V}_1| \cdot n / (i \cdot (n - i))$ . To see this note that if we restrict our attention to the subgraph of  $\mathcal{G}$  spanned by instances in  $\mathcal{V}_1 \cup \mathcal{T}_i$  then we have a bipartite graph where each instance in  $\mathcal{V}_1$  has the same degree  $n$  (or  $n/2$  if  $i = n/2$ ) and each instance in  $\mathcal{T}_i$  has the same degree  $i \cdot (n - i)$ . Therefore, the total number of edges incident on  $\mathcal{V}_1$  is  $\leq |\mathcal{V}_1| \cdot n$  and those incident on  $\mathcal{T}_i$  is  $|\mathcal{T}_i| \cdot i \cdot (n - i)$ . Since the number of edges should be the same counted from either side, we get  $|\mathcal{T}_i| \leq |\mathcal{V}_1| \cdot n / (i \cdot (n - i))$ . Now we finish the proof of the lemma with the following calculation:

$$|\mathcal{V}_2| = \sum_{i=3}^{n/2} |\mathcal{T}_i| \leq \sum_i \frac{n}{i \cdot (n - i)} \cdot |\mathcal{V}_1| = |\mathcal{V}_1| \cdot O(\log n) \quad \blacktriangleleft$$

## 4 Lower Bounds in the KT-1 Model

Our lower bounds in the KT-1 model are inspired by the work of Hajnal et al. [14], which is concerned with 2-party communication complexity of several graph problems, including CONNECTIVITY. In their setup [14], the input graph  $G = (V, E)$  is *edge-partitioned* among Alice and Bob in such a way that both parties know  $V$  and Alice and Bob respectively know edge sets  $E_A$  and  $E_B$ , where  $(E_A, E_B)$  forms a partition of  $E$ . One simple deterministic protocol that solves CONNECTIVITY in this setup is this: Alice sends all the connected components induced by  $E_A$  to Bob, who can determine if  $G$  is connected. The worst case communication complexity of this protocol is  $O(n \log n)$ . Via reduction from PARTITION, Hajnal et al. [14] show that there exists a family of input graphs such that for *any equal sized* edge partition, the communication complexity of CONNECTIVITY is  $\Omega(n \log n)$ .

It does not seem possible to reduce from this edge-partitioned version of 2-party CONNECTIVITY to CONNECTIVITY in the KT-1 model because KT-1 algorithms are vertex-centric and Alice and Bob may not hold all the edges they need to simulate vertices executing a

KT-1 algorithm. We resolve this issue by designing a new reduction, from PARTITION to a vertex-partition version of 2-party CONNECTIVITY. In the Hajnal et al. [14] reduction, PARTITION is reduced to CONNECTIVITY on a family of dense graphs. Motivated by our KT-0 lower bound for CONNECTIVITY for the TWOCYCLE problem, we are interested in deriving a KT-1 CONNECTIVITY lower bound for a *sparse* class of graphs as well. In what follows, we extend the reduction of Hajnal et al. from PARTITION to CONNECTIVITY in two important ways: (i) we reduce to a vertex-partitioned version of CONNECTIVITY and (ii) we reduce to a sparse special case of CONNECTIVITY that we call the MULTICYCLE problem, in which the input is either a single cycle or two or more cycles, each having length at least 4.

#### 4.1 A Special Case of the Partition Problem

In order to establish a lower bound for MULTICYCLE, we now consider a special case of the 2-party PARTITION problem, which we call TWOPARTITION. The input to TWOPARTITION consists of partitions  $P_A$  and  $P_B$  of  $[n]$ , for even  $n$ , such that each part in  $P_A$  and  $P_B$  has exactly two elements in it. We will now use a linear algebraic argument to show that there is an  $\Omega(n \log n)$  deterministic lower bound on this special case of PARTITION also. The 0-1 matrix  $E^n$  associated with this problem is a sub-matrix of the matrix  $M^n$  where  $M^n(i, j) = 1$  if  $P_i \vee P_j = 1$  and  $M^n(i, j) = 0$  otherwise (see Section 2). The matrix  $E^n$  has dimension  $r \times r$  where  $r = n!/(2^{n/2} \cdot (n/2)!)$ . This fact follows from a simple counting argument. In the following theorem, we show that this sub-matrix  $E^n$  has full rank.

► **Lemma 14.**  $rank(E^n) = r$  where  $r = n!/(2^{n/2} \cdot (n/2)!)$ .

**Proof.** We will prove a more general observation: every sub-matrix  $A_S$  of a full rank  $d \times d$  matrix  $A$  formed by choosing a subset  $S$  of the rows and the corresponding columns has rank  $s$  where  $s = |S|$ . In other words, for all  $S$ ,  $A_S$  is a full rank  $s \times s$  matrix.

Let  $B$  be a  $d \times d$  diagonal matrix where  $B(i, i) = 1$  if  $i \in S$  and  $B(i, i) = 0$  if  $i \notin S$ . It is easy to see that  $rank(B) = |S| = s$ . Using basic properties of rank,  $rank(AB) \leq rank(B) \leq s$  and by Sylvester's rank inequality<sup>4</sup>,  $rank(AB) \geq rank(A) + rank(B) - d = d + s - d = s$ .

Therefore,  $rank(AB) = s$  which means that some minor of  $AB$  having dimension  $s$  needs to be of full rank. The only such candidate is the minor corresponding to the matrix  $A_S$  because all other minors of dimension  $s$  either have an all zero row or all zero column. Therefore,  $A_S$  has full rank.

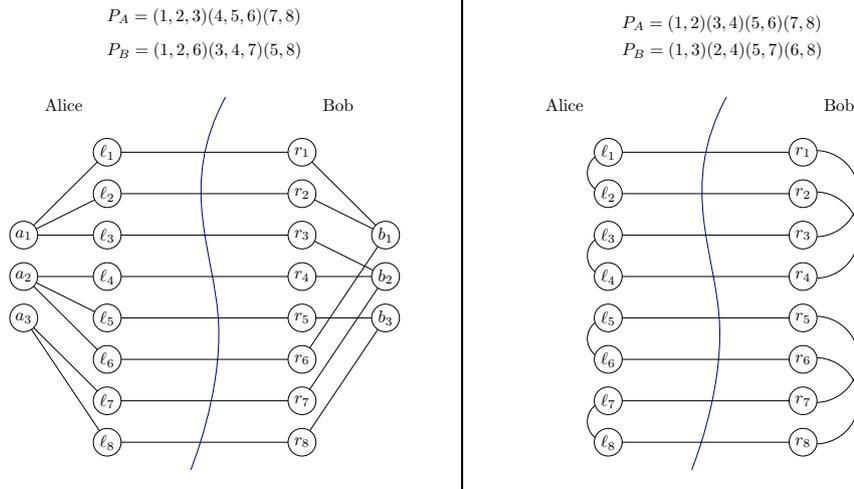
Now  $E^n$  is a submatrix of  $M^n$  where the rows and columns correspond to partitions of  $[n]$  such that each part has exactly two elements in it. Therefore, the lemma follows since  $M^n$  has full rank. ◀

By using Stirling's approximation, it can be verified that  $r = 2^{\Theta(n \log n)}$ . Then, by the rank bound and Lemma 1.28 of [23] we get the following corollary.

► **Corollary 15.** *The deterministic 2-party communication complexity of TWOPARTITION is  $\Omega(n \log n)$*

We describe our reductions in the next two subsections. In section 4.2, we reduce the PARTITION (TWOPARTITION) problem to the vertex partitioned 2-party CONNECTIVITY (2-party MULTICYCLE) problem and in section 4.3, we reduce the 2-party CONNECTIVITY (2-party MULTICYCLE) problem to CONNECTIVITY (MULTICYCLE) in the KT-1 model.

<sup>4</sup> For any two  $n \times n$  matrices  $A, B$ ,  $rank(AB) \geq rank(A) + rank(B) - n$ . We can prove this inequality by applying the rank-nullity theorem to the inequality  $null(AB) \leq null(A) + null(B)$ .



■ **Figure 2** The figure on the left illustrates the reduction from PARTITION to 2-party CONNECTIVITY and the figure on the right illustrates the reduction from TWOPARTITION to 2-party MULTICYCLE. The vertices  $a_4, \dots, a_8$  that are connected to  $\ell_* = \ell_8$  and  $b_4, \dots, b_8$  connected to  $r_* = r_8$  are not shown in the left figure.

## 4.2 Reductions from Partition and TwoPartition

Here we present two reductions, first from PARTITION to 2-party CONNECTIVITY and next from TWOPARTITION to 2-party MULTICYCLE. Alice is given a partition  $P_A = (S_1, S_2, \dots, S_n)$  over the ground set  $[n]$  where  $S_i$  is the  $i^{\text{th}}$  part of  $P_A$ , which could possibly be empty if  $P_A$  has fewer than  $i$  parts. Similarly, Bob is given a partition  $P_B = (S'_1, S'_2, \dots, S'_n)$ . They construct a graph  $G(P_A, P_B)$  as follows: Alice creates vertex sets  $A = \{a_1, \dots, a_n\}$  and  $L = \{\ell_1, \dots, \ell_n\}$  whereas Bob creates the vertex sets  $R = \{r_1, \dots, r_n\}$  and  $B = \{b_1, \dots, b_n\}$ . Alice and Bob add edges  $(\ell_i, r_i)$  for  $i \in [n]$ , independent of  $P_A$  and  $P_B$ . Alice adds edges between  $A$  and  $L$  that induce the partition  $P_A$  on  $L$ . That is, for every  $S_i \in P_A$ , Alice adds edges  $(a_i, \ell_j)$  for all  $j \in S_i$ . There will be some vertices in  $A$  that are not connected to any vertex, so Alice just adds an edge between these vertices and an arbitrary vertex  $\ell_* \in L$ . Bob similarly adds edges between the sets  $B$  and  $R$ . See Figure 2.

If  $P_A$  and  $P_B$  are instances of TWOPARTITION, that is, each part of  $P_A$  and  $P_B$  is of size exactly two, then we can modify the construction of  $G(P_A, P_B)$  by getting rid of the sets  $A$  and  $B$ . Note that in this case  $P_A = (S_1, S_2, \dots, S_{n/2})$  and  $P_B = (S'_1, S'_2, \dots, S'_{n/2})$  where each  $S_i$  and  $S'_i$  has size exactly two. If  $\{i, j\} \in P_A$  then Alice creates an edge between  $\ell_i$  and  $\ell_j$  and Bob does the same with  $R$  for every pair in  $P_B$ . With this modified construction, each vertex in  $G(P_A, P_B)$  has degree exactly 2 and therefore, every connected component of  $G(P_A, P_B)$  will be a cycle. See Figure 2.

The following theorem encapsulates a crucial property of the graph  $G(P_A, P_B)$  which implies the correctness of our reductions.

► **Theorem 16.** *If  $P_A$  and  $P_B$  are instances of PARTITION (or TWOPARTITION), then the partition induced by the connected components of  $G(P_A, P_B)$  on the vertices in  $L$  and  $R$  corresponds to the partition  $P_A \vee P_B$ .*

**Proof.** Call two elements  $a$  and  $b$  *reachable* from each other if there exists a sequence of distinct elements  $e_0, e_1, \dots, e_t, 1 \leq t \leq n$  such that  $e_0 = a$ ,  $e_t = b$  and each pair  $(e_i, e_{i+1})$  either belongs to the same part of  $P_A$  or the same part of  $P_B$ . Any partition in which all reachable elements are in the same part have both  $P_A$  and  $P_B$  as refinements.

We claim that two elements belong to the same part of  $P_A \vee P_B$  if and only if they are reachable from each other. The backward direction is true because  $P_A$  and  $P_B$  are both refinements of  $P_A \vee P_B$ . The forward direction is true because if  $a$  and  $b$  are not reachable from each other but still belong to the same part  $S$  of  $P_A \vee P_B$  then we can refine the part  $S$  to be  $S_a, S_b$  where  $S_a$  is the set of all elements in  $S$  that are reachable from  $a$  and  $S_b$  is the set of all elements in  $S$  that are reachable from  $b$ . It is easy to see that  $S_a$  and  $S_b$  are disjoint. Let  $P'$  be the partition  $P_A \vee P_B$  where  $S$  is further refined to be  $S_a, S_b, S \setminus (S_a \cup S_b)$ . Note that with this further refinement of  $S$ , we still have the property that all pairs of reachable elements belong to the same part of  $P'$ . This means both  $P_A$  and  $P_B$  still remain refinements of the  $P'$  which contradicts the minimality of the join.

The theorem follows by observing that  $i$  and  $j$  are reachable from each other if and only if there is a path from  $\ell_i$  to  $\ell_j$  (and consequently from  $r_i$  to  $r_j$ ) in  $G(P_A, P_B)$ . ◀

### 4.3 Reductions from 2-party Connectivity and MultiCycle

We now show reductions from 2-party CONNECTIVITY to CONNECTIVITY in the KT-1 model and from 2-party MULTICYCLE to MULTICYCLE in the KT-1 model. Given an  $r$ -round KT-1 algorithm  $\mathcal{A}$ , Alice and Bob will simulate the algorithm with  $G(P_A, P_B)$  as the input graph. Alice hosts vertices in  $A \cup L$  and Bob hosts vertices in  $B \cup R$ . For  $1 \leq i \leq n$ , the IDs of vertices  $a_i, \ell_i, r_i$ , and  $b_i$  are  $i, n + i, 2n + i$ , and  $3n + i$  respectively. So both parties know the ID's of all vertices as well as the ID's of neighbors of all hosted vertices in  $G(P_A, P_B)$  and hence, the initial knowledge of hosted vertices.

In order to simulate round  $t$  of  $\mathcal{A}$ , Alice and Bob need to compute the states of all hosted vertices after round  $t$  of  $\mathcal{A}$ . The state of a vertex  $v$  after round  $t$  depends on the initial knowledge and the transcript  $\tau(v, t)$  of  $v$ . Assume that Alice and Bob know the states of all the vertices they host after round  $t - 1$ . Alice and Bob send a message from  $\{0, 1, \perp\}^{2n}$  to each other. These messages denote the characters their hosted vertices broadcast in round  $t$ , in increasing order of ID. Therefore, they know the sender ID of a character from the position of the character in the message. This enables Alice and Bob to compute the transcript  $\tau(v, t)$  and hence the state after round  $t$  of all hosted vertices  $v$ .

Therefore, in simulating each round, Alice and Bob exchange exactly  $O(n)$  bits with each other and the total communication complexity of the protocol is  $O(rn)$ . If  $\mathcal{A}$  solves the CONNECTIVITY or MULTICYCLE problems, then using Corollaries 4 and 15 respectively and Theorem 16, we obtain the following result.

► **Theorem 17.** *The round complexity of a deterministic algorithm for solving the CONNECTIVITY and MULTICYCLE problems in the KT-1 model is  $\Omega(\log n)$ .*

### 4.4 Information-theoretic Lower Bound for ConnectedComponents

Já Já [17] proves a lower bound for 2-party CONNECTEDCOMPONENTS and points out that his techniques may not work for decision problems, indicating that it might be easier to prove lower bounds for CONNECTEDCOMPONENTS. This motivates us to consider the CONNECTEDCOMPONENTS problem as a lower bound candidate, closely related to CONNECTIVITY, but for which we may be able to prove an  $\Omega(\log n)$  lower bound in the KT-1 model, *even for constant-error Monte Carlo algorithms*. It turns out that we are able to prove this result

by combining the reductions described in the previous section with information-theoretic techniques. We first define the 2-party problem `PARTITIONCOMP` which is closely related to `PARTITION`, but requires an output with a large representation. As in `PARTITION`, Alice and Bob are respectively given set partitions  $P_A$  and  $P_B$  of  $[n]$  and at the end of the communication protocol for `PARTITIONCOMP`, Alice and Bob are required to output the join  $P_A \vee P_B$ . From Theorem 16, we get that if there is a  $t$ -round,  $\epsilon$ -error Monte Carlo algorithm  $\mathcal{A}$  for `CONNECTEDCOMPONENTS` in the `KT-1` model, then there is an  $\epsilon$ -error Monte Carlo protocol that solves `PARTITIONCOMP` with communication complexity  $t \cdot n$ .

Consider the following distribution over inputs of `PARTITIONCOMP`: Alice's input  $P_A$  is chosen uniformly at random from the set of all partitions and Bob's partition is fixed to be the finest partition, i.e.,  $P_B = (1)(2)(3) \dots (n)$ . With  $P_B$  fixed in this manner,  $P_A \vee P_B = P_A$  and at the end of the protocol Bob learns  $P_A$ . Since  $P_A$  is chosen from the uniform distribution, its initial entropy is  $\Theta(n \log n)$  since the support of the distribution has size  $2^{\Theta(n \log n)}$ . Therefore Bob will learn a lot of information by the end of the protocol. This idea is formalized in the proof of the following theorem. This proof also has to deal with the complication that the protocol has constant error probability.

► **Theorem 18.** *For any constant  $0 < \epsilon < 1$ , the round complexity of an  $\epsilon$ -error randomized Monte Carlo algorithm that solves the `CONNECTEDCOMPONENTS` problem in the `KT-1` version of the `BCC(1)` model is  $\Omega(\log n)$ .*

**Proof.** Using Yao's minimax theorem (Theorem 2) we can assume that all protocols are deterministic but are allowed to make an error on  $\epsilon$ -fraction of the input, weighted by  $\mu$ . Although appealing to Yao's theorem is not necessary, it allows us to simplify the exposition. Let  $\Pi$  denote the transcript of a 2-party protocol that solves `PARTITIONCOMP` and let  $|\Pi|$  denote the length of the longest transcript produced by  $\Pi$  on any input. We know that

$$|\Pi| \geq H(\Pi(P_A, P_B)) \geq I(\Pi(P_A, P_B); P_A, P_B) = I(P_A, P_B; \Pi(P_A, P_B)) = I(P_A; \Pi(P_A, P_B))$$

where the last equality follows from the fact that  $P_B$  is fixed according to  $\mu$ . From the definition of mutual information,  $I(P_A; \Pi(P_A, P_B)) = H(P_A) - H(P_A | \Pi(P_A, P_B))$ . Alice's input  $P_A$  is uniformly distributed among all  $B_n = 2^{\Theta(n \log n)}$  set partitions according to the hard distribution  $\mu$ . Therefore  $H(P_A) = \Theta(n \log n)$ . Let  $B$  be the set of protocol transcripts that produce an error on the input  $P_A, P_B$ . If  $\Pi(P_A, P_B) \notin B$  then  $H(P_A | \Pi(P_A, P_B)) = 0$  since the output of the protocol is  $P_A \vee P_B = P_A$ . We are guaranteed that  $\Pr[\Pi(P_A, P_B) \in B] \leq \epsilon$ . Therefore, the second term can be bounded as follows.

$$\begin{aligned} H(P_A | \Pi(P_A, P_B)) &= \sum_{\pi} \Pr[\Pi(P_A, P_B) = \pi] H(P_A | \Pi(P_A, P_B) = \pi) \\ &= \sum_{\pi \in B} \Pr[\Pi(P_A, P_B) = \pi] H(P_A | \Pi(P_A, P_B) = \pi) \leq \epsilon H(P_A) \end{aligned}$$

Where the last inequality follows from the fact that  $H(X|Y) \leq H(X)$  for any  $X, Y$ . This implies  $I(P_A; \Pi(P_A, P_B)) = \Omega(n \log n)$  which proves that any  $\epsilon$ -error randomized protocol that solves the `PARTITIONCOMP` problem has communication complexity of  $\Omega(n \log n)$ . This in turn implies that  $t = \Omega(\log n)$  which proves the theorem. ◀

---

## References

- 1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling views: A new lower bound technique for distributed computations under congestion. *CoRR*, 2017. [arXiv:1711.01623](https://arxiv.org/abs/1711.01623).

- 2 Baruch Awerbuch, Oded Goldreich, David Peleg, and Ronen Vainish. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990. doi:10.1145/77600.77618.
- 3 Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 315–324, 2015. doi:10.1145/2767386.2767421.
- 4 Florent Becker, Antonio Fernández Anta, Ivan Rapaport, and Eric Rémila. The effect of range and bandwidth on the round complexity in the congested clique model. In *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, pages 182–193, 2016. doi:10.1007/978-3-319-42634-1\_15.
- 5 Keren Censor-Hillel, Seri Khoury, and Ami Paz. Quadratic and near-quadratic lower bounds for the CONGEST model. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 10:1–10:16, 2017. doi:10.4230/LIPIcs.DISC.2017.10.
- 6 Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional lower bounds for adaptive massively parallel computation. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '20*, page 141–151, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3350755.3400230.
- 7 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.
- 8 Artur Czumaj and Christian Konrad. Detecting cliques in congest networks. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, July 2018. URL: <http://wrap.warwick.ac.uk/106950/>.
- 9 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC '11*, pages 363–372, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993686.
- 10 Thomas A. Dowling and Richard M. Wilson. Whitney number inequalities for geometric lattices. *Proceedings of the American Mathematical Society*, 47(2):504–504, 1975. doi:10.1090/s0002-9939-1975-0354422-3.
- 11 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing, PODC '14*, pages 367–376, New York, NY, USA, 2014. ACM. doi:10.1145/2611462.2611493.
- 12 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA '18*, pages 153–162, New York, NY, USA, 2018. ACM. doi:10.1145/3210377.3210401.
- 13 Mohsen Ghaffari and Merav Parter. MST in Log-Star Rounds of Congested Clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 19–28, 2016. doi:10.1145/2933057.2933103.
- 14 András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 186–191, 1988. doi:10.1145/62212.62228.
- 15 James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and mst. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC '15*, pages 91–100, New York, NY, USA, 2015. ACM. doi:10.1145/2767386.2767434.
- 16 Stephan Holzer and Nathan Pinsker. Approximation of distances and shortest paths in the broadcast congest clique. In *19th International Conference on Principles of Distributed*

- Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, pages 6:1–6:16, 2015. doi:10.4230/LIPIcs.OPODIS.2015.6.
- 17 Joseph Já Já. The vlsi complexity of selected graph problems. *J. ACM*, 31(2):377–391, March 1984. doi:10.1145/62.70.
  - 18 Tomasz Jurdzinski, Krzysztof Lorys, and Krzysztof Nowicki. Communication complexity in vertex partition whiteboard model. In *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, pages 264–279, 2018. doi:10.1007/978-3-030-01325-7\_24.
  - 19 Tomasz Jurdzinski and Krzysztof Nowicki. Brief announcement: On connectivity in the broadcast congested clique. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 54:1–54:4, 2017. doi:10.4230/LIPIcs.DISC.2017.54.
  - 20 Tomasz Jurdziński and Krzysztof Nowicki. Mst in  $o(1)$  rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 2620–2632, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3174304.3175472>.
  - 21 E. Korach, S. Moran, and S. Zaks. The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. *SIAM J. Comput.*, 16(2):231–236, April 1987. doi:10.1137/0216019.
  - 22 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.
  - 23 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.
  - 24 Pedro Montealegre and Ioan Todinca. Brief announcement: Deterministic graph connectivity in the broadcast congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC '16*, pages 245–247, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933066.
  - 25 Pedro Montealegre and Ioan Todinca. Deterministic graph connectivity in the broadcast congested clique. *CoRR*, abs/1602.04095, 2016. URL: <http://arxiv.org/abs/1602.04095>.
  - 26 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '19*, page 1844–1860, USA, 2019. Society for Industrial and Applied Mathematics.
  - 27 Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*, pages 1–17, 2017. doi:10.1007/978-3-319-69084-1\_1.
  - 28 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
  - 29 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and circuits: (on lower bounds for modern parallel computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 1–12, 2016. doi:10.1145/2935764.2935799.
  - 30 D.J.A. Welsh. *Matroid theory*. Dover Publications, 2010. URL: <http://www.worldcat.org/oclc/319491697?referer=xid>.
  - 31 A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, October 1977. doi:10.1109/SFCS.1977.24.



# Fully Dynamic Sequential and Distributed Algorithms for MAX-CUT

Omer Wasim 

Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA  
wasim.o@northeastern.edu

Valerie King

Department of Computer Science, University of Victoria, Canada  
val@uvic.ca

---

## Abstract

This paper initiates the study of the MAX-CUT problem in fully dynamic graphs. Given a graph  $G = (V, E)$ , we present deterministic fully dynamic distributed and sequential algorithms to maintain a cut on  $G$  which *always* contains at least  $\frac{|E|}{2}$  edges in sublinear update time under edge insertions and deletions to  $G$ . Our results include the following deterministic algorithms: i) an  $O(\Delta)$  *worst-case* update time sequential algorithm, where  $\Delta$  denotes the maximum degree of  $G$ , ii) the first fully dynamic distributed algorithm taking  $O(1)$  rounds and  $O(\Delta)$  total bits of communication per update in the Massively Parallel Computation (MPC) model with  $n$  machines and  $O(n)$  words of memory per machine. The aforementioned algorithms require at most *one* adjustment, that is, a move of one vertex from one side of the cut to the other.

We also give the following fully dynamic sequential algorithms: i) a deterministic  $O(m^{1/2})$  amortized update time algorithm where  $m$  denotes the maximum number of edges in  $G$  during any sequence of updates and, ii) a randomized algorithm which takes  $\tilde{O}(n^{2/3})$  worst-case update time when edge updates come from an oblivious adversary.

**2012 ACM Subject Classification** Theory of computation; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** data structures, dynamic graph algorithms, approximate maximum cut, distributed computing, parallel computing

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.33

**Funding** This research was completed while the first author was a MSc candidate at the University of Victoria. Both authors were funded by a NSERC Discovery Grant.

**Acknowledgements** The authors thank Hung Le for useful discussions and an anonymous reviewer for helpful comments.

## 1 Introduction

A fully dynamic graph algorithm is a data structure to maintain a property of a graph under an arbitrary sequence of edge insertions and deletions. The goal is to update the graph in less time than the best static algorithm which computes the property from scratch. A *fully dynamic* graph algorithm may incur preprocessing time, after which it is able to answer queries regarding the maintained property. Research in this area has focused mostly on dynamic variants of well-known problems such as connectivity [42, 26, 30, 15], minimum spanning trees [24, 26, 47], minimum cut [45], etc., all of which admit polynomial time exact algorithms in the static setting.

Following the seminal work of Onak and Rubinfeld [40] in which fully dynamic algorithms for maintaining constant factor approximations of maximum matching (and vertex cover) were presented, research in dynamic algorithms has broadened to include approximate versions of NP-hard problems. Some natural directions arising in this setting include the design



© Omer Wasim and Valerie King;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 33; pp. 33:1–33:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of dynamic algorithms to maintain an approximate solution in sublinear update time and the study of approximability-time trade-off. A list of approximate versions of NP-hard problems investigated in the dynamic setting includes vertex cover [5, 43, 9, 38], set-cover [22], dominating set [25], graph coloring [8], facility location [21] and maximum independent set [23, 3, 4].

In this paper, we initiate the study of the MAX-CUT problem in fully dynamic graphs and pose the question of whether there exist sublinear update time algorithms. Another parameter we look at is the *adjustment cost*, which is defined in a dynamic graph problem as the amount of changes to the maintained solution per update. In the case of MAX-CUT, it is the number of vertices which move from one subset of the cut to the other.

MAX-CUT is one of the fundamental NP-hard problems [32] which continues to be widely studied. Some of its concrete applications arise in the design of integrated circuits [12], communication networks [14] and statistical physics [41]. It also models a standard 2-clustering objective for partitioning a graph such that the number of inter-cluster edges is maximized.

Let  $G = (V, E)$  be an undirected, unweighted graph  $G = (V, E)$  with  $n = |V|, m = |E|$ . A cut  $C$  is a partition of the vertex set  $V$ , and denoted by  $C = (S, \bar{S})$ , where  $S, \bar{S} \subseteq V$  and  $\bar{S} = V \setminus S$ . The cut-set  $E(S, \bar{S})$  of  $C = (S, \bar{S})$  is the set of all edges which have exactly one endpoint in  $S$ . A cut edge of  $C$  is an edge contained in the cut-set  $E(C) = E(S, \bar{S})$ . A maximum cut of  $G$  is a cut whose cut-set is largest among cut-sets for all possible cuts, i.e.  $\text{MAX-CUT}(G) = \arg \max_{C=(S, \bar{S}), S \subseteq V} |E(S, \bar{S})|$ , where  $|E(S, \bar{S})|$  denotes the number of cut edges. We say a cut is  $t$ -respecting if  $|E(C)| \geq t|E|$ . Note that the cut-set of a  $t$ -respecting cut contains a  $t$  fraction of all edges, regardless of the size of the largest cut-set. Let  $OPT$  denote the size of the largest cut-set. A cut is  $t$ -approximate if  $|E(C)| \geq t \cdot OPT$  and a  $t$ -approximation algorithm for MAX-CUT yields a  $t$ -approximate cut. It follows that a  $t$ -respecting cut is *always* a  $t$ -approximate cut but not vice-versa. This distinction can be appreciated in the case of  $K_{2n}$ , the complete graph on  $2n$  vertices where a maximum cut is any cut  $C = (S, \bar{S})$  where  $|S| = n$ . For large  $n$ , the size of the cut-set of a  $\frac{1}{2}$ -respecting cut can be nearly twice the size of a  $\frac{1}{2}$ -approximate cut. Throughout this paper, we let  $[k]$  to denote  $\{1, 2, \dots, k\}$ ,  $\Delta$  to be the maximum degree of  $G$  and  $\tilde{O}$  to hide a  $O(\text{polylog}(n))$  factor.

The Massively Parallel Computation (MPC) model was introduced by Karloff et al. [31] and later refined in [20, 6, 1] as a theoretical framework for large scale parallel processing settings such as those in [48, 17]. There are  $\mu$  machines with  $S$  words of memory each, which solve a problem by synchronously communicating over an all-to-all communication network (i.e. a complete network). Initially, input data of size  $N$  (which is  $O(m + n)$  in the case of a graph problem) are distributed across these machines. It is desirable to have  $\mu$  and  $S$  to be  $O(N^{1-\epsilon})$  for some  $\epsilon > 0$  and the message size is limited to  $O(S)$  bits. In each round, every machine can: i) receive messages of the previous round from other machines ii) do local polynomially bounded computation (i.e. taking  $\text{poly}(S)$  space and time) without additional communication and iii) send messages to other machines which are received in the next round. The complexity of a MPC algorithm to solve a problem is determined by 3 parameters: i) the number of rounds of communication, ii) the size of the memory per machine and iii) the total amount of communication per round. Typically, MPC algorithms for graph problems use  $O(n)$  machines,  $\tilde{O}(n)$  memory per machine and take  $\tilde{O}(1)$  rounds of communication.

## 1.1 Previous Work

**Static sequential algorithms for  $\frac{1}{2}$ -respecting cuts.** A simple randomized algorithm, hereafter referred to as Randomized Max-Cut obtains a  $\frac{1}{2}$ -respecting cut  $C = (S, \bar{S})$  in expectation by placing each vertex independently in  $S$  or  $\bar{S}$  with probability  $\frac{1}{2}$ . Any edge  $e = \{u, v\}$  is a cut edge of  $C$  with probability  $\frac{1}{2}$ , implying the result. Randomized Max-Cut can be derandomized using the method of conditional expectation or pairwise independence.

Johnson's folklore algorithm [28] hereafter referred to as Greedy Max-Cut, which finds a  $\frac{1}{2}$ -respecting cut can be viewed as derandomized version of Randomized Max-Cut using the method of conditional expectation. Given  $G = (V, E)$ , where  $V = \{v_1, v_2, \dots, v_n\}$  it starts with  $S = \{v_1\}, \bar{S} = \emptyset$ . Each successive vertex  $v_j$ , where  $j \geq 2$  is added to  $S$  or  $\bar{S}$  depending on which contains fewer of its neighbors  $v_i$ , where  $i < j$ . Thus, at least half of all edges of the form  $\{v_j, v_i\}$  where  $i < j$  are contained in the resulting cut. Since each vertex and edge is encountered once, the running time of Greedy Max-Cut is  $O(m + n)$ .

Randomized Max-Cut can also be derandomized using the idea of pairwise independence [37]. For a set  $S$ , let  $\mathcal{P}(S)$  denote the power set of  $S$ . We first note that one can get a  $\frac{1}{2}$ -respecting cut (in expectation) which uses only  $k = \lceil \log n \rceil$  independent random bits. The idea is to construct a one-to-one function  $f : V \rightarrow \mathcal{P}([k])$  and choosing  $R$  to be a uniformly random subset of  $[k]$ . It can be shown that the cut  $C = (S, \bar{S})$  where  $S = \{v \mid |f(v) \cap R| \text{ is even}\}$  and  $\bar{S} = \{v \mid |f(v) \cap R| \text{ is odd}\}$  is  $\frac{1}{2}$ -respecting in expectation. Enumerating all the  $2^k = O(n)$  possibilities for  $R$ , and taking the cut which maximizes  $|E(S, \bar{S})|$  yields a  $\frac{1}{2}$ -respecting cut. For a fixed  $R$ , the time to compute  $C$  is  $O(nk)$  while determining the size of  $C$ 's cut-set takes  $O(m)$  time giving a total time of  $O(n^2 \log n + mn)$ . While this algorithm isn't better in terms of running time as compared to Greedy Max-Cut, it has the advantage of being parallelizable.

**Static distributed algorithms for  $\frac{1}{2}$ -respecting cuts.** We observe that the algorithm obtained by derandomizing Randomized Max-Cut via pairwise independence can be used to compute a  $\frac{1}{2}$ -respecting cut in  $O(1)$  rounds in the MPC model of computation with  $n$  machines and  $\Theta(n)$  memory per machine. We assume there exists a fixed coordinator machine. Given  $f$ , each machine corresponds to a vertex  $v$ , and stores  $f(v)$  along with the list of  $v$ 's neighbors and  $R \subseteq [k]$  which is fixed. In the first round, each machine first computes the count of the number of edges its corresponding vertex is incident to in the cut obtained by considering the  $i^{\text{th}}$  choice of  $R$  where  $i \in [n]$ . Then each machine sends the  $i^{\text{th}}$  count to machine  $i$ . In the next round all machines send these counts to the coordinator, which chooses a  $\frac{1}{2}$ -respecting cut and informs all other machines. Thus, at the end of the third round, each machine is able to output the position of its corresponding vertex in the  $\frac{1}{2}$ -respecting cut. The total amount of communication is bounded by  $O(n^2 \log n)$  bits.

A similar adaptation of Greedy Max-Cut in the MPC model with  $n$  machines and  $\Theta(n)$  memory per machine takes  $n$  rounds of communication and  $O(n\Delta)$  total communication.

The only *deterministic* distributed algorithm to compute a  $\frac{1}{2}$ -respecting cut that we are aware of was presented by Censor-Hillel et al. [11] which takes  $\tilde{O}(\Delta + \log^* n)$  rounds and  $\Omega(\Delta^2)$  messages in the *CONGEST* model. Their algorithm can be adapted to the Congested-Clique setting with the same round and message complexity.

**Approximation Algorithms for MAX-CUT.** We briefly survey the relevant literature on approximation algorithms for MAX-CUT in the static setting. Goemans and Williamson (1994) used a semidefinite programming (SDP) relaxation [19] and randomized rounding to yield a 0.878-approximation to MAX-CUT. This polynomial-time algorithm runs in

super-linear time using state-of-the art numerical methods for solving a semidefinite program. Khot et al. showed that MAX-CUT is hard to approximate better than 0.878 [35] under the Unique Games Conjecture [34].

Arora and Kale [2] presented a primal dual (SDP-based)  $(0.878 - \epsilon)$ -approximation algorithm which runs in  $\tilde{O}(m)$  time for  $d$  regular graphs with high probability where the running time depends inversely on  $\epsilon$ . Trevisan later presented a 0.53-approximation algorithm for MAX-CUT utilizing spectral techniques [46] whose analysis was improved to 0.62 by Soto [44]. In the same paper, Trevisan showed that the primal dual SDP-based algorithm of [2] can be made to run in  $\tilde{O}(m)$  time for any degree, via a linear time reduction to reduce the maximum degree to  $O(\text{polylog}(n))$ . By using the algorithm of Arora and Kale [2] together with the rounding scheme of Charikar and Wirth [13], we note that in graphs in which the size of the optimal cut is  $(\frac{1}{2} + \epsilon)|E|$ , one can get a  $(\frac{1}{2} + \Omega(\frac{\epsilon}{\log(1/\epsilon)}))$ -respecting cut in  $\tilde{O}(m)$  time. However, when  $\epsilon = O(\frac{1}{n})$  (as in the case of  $K_{2n}$ ) and a  $\frac{1}{2}$ -respecting cut is desired (instead of a  $\frac{1}{2}$ -approximate cut) this can take  $\Omega(mn)$  time.

Kale and Seshadhri [29] presented a combinatorial algorithm based on the spectral method [46] which uses random walks to give a  $(0.5 + \epsilon)$ -approximation with running time depending on  $\epsilon$ . For  $\epsilon = 0.0155$ , the running time is  $\tilde{O}(n^2)$ . As the running time increases, the approximation ratio converges to the spectral algorithm of Trevisan [46].

## 1.2 The Fully Dynamic Model

In this paper, we seek to maintain a  $\frac{1}{2}$ -respecting cut in sublinear update time and handle meaningful queries such as determining whether an edge is in the cut-set, the size of vertex partitions and the cut-set in *constant time*. We define the Fully Dynamic Max-Cut problem as follows:

► **Problem 1** (Fully Dynamic MAX-CUT). Starting with a graph  $G = (V, E)$  on  $n$  vertices and an empty edge set  $E$ , maintain a  $\frac{1}{2}$ -respecting cut  $C = (S, \bar{S})$  for  $G$  under edge insertions and deletions to  $E$  such that queries of the following form can be handled in constant time: i) Is the edge  $\{v_i, v_j\}$  contained in the cut-set  $E(S, \bar{S})$ ? ii) What is the size of the cut-set,  $E(C)$ ? iii) What are the sizes of  $S$  and  $\bar{S}$ ?

Our goal is to update  $C$  in  $o(m + n)$  time to fare better than running Greedy Max-Cut after every update and we require that answers to all queries between any two updates must be consistent with respect to the maintained cut  $C$ .

In the fully dynamic MPC model that we consider in this paper, we start with a graph  $G = (V, E)$  on  $n$  vertices and  $m$  edges. Let  $N = O(m + n)$ . We use a coordinator machine which can be selected in a single round: machines send their ID's to all other machines and the coordinator is selected to be the machine with ID larger than all ID's it receives. There are a total of  $n$  machines each with  $\Theta(n)$  memory and the goal is to maintain a  $\frac{1}{2}$ -respecting cut in  $O(1)$  rounds per edge update and  $O(n)$  total communication per round. Each machine corresponds to a vertex of  $G$  and stores the edges incident to it. After any update  $\{u, v\}$  to the graph, the machines corresponding to  $u$  and  $v$  are informed of the update. In our model, we insist on algorithms which make few adjustments to the maintained cut. We note that there are other fully dynamic MPC models that have been studied very recently such as in [27, 18, 39]. Our dynamic algorithm in the MPC model requires at most one adjustment. Ensuring this is easier in the case of problems such as maximal matching where only the neighborhood of endpoints of the updated edge needs to be examined per update. In our case, this may not always be the case (see Theorem 9).

Attaining a *deterministic worst-case* update time (i.e. without randomization or amortization) is an important objective in the design of dynamic algorithms. For the seminal problem of dynamic connectivity, deterministic algorithms beating  $O(\sqrt{n})$  update time [15, 33] were only recently discovered after decades. Another example is the maximal independent set problem for which known deterministic algorithms [3, 23] only achieve a sublinear (in  $m$ ) amortized update time and polylogarithmic update time algorithms are yet to be discovered.

An event happens with high probability (w.h.p) if its probability is  $1 - \frac{1}{n^c}$  for any  $c > 0$ . For our randomized algorithm, we assume that updates come from an oblivious adversary. This is a standard assumption used in the design of many randomized dynamic algorithms. An oblivious adversary is one which cannot choose updates adaptively in response to the answers returned by queries. Thus, updates to the graph can be assumed to be fixed in advance. We assume the existence of an oracle which randomly labels each vertex uniquely using a number in  $\{1, \dots, n\}$ , and to which the adversary is oblivious. This is only used in the algorithm of Theorem 6. We seek to maintain a  $\frac{1}{2}$ -respecting cut *exactly* or w.h.p.

### 1.2.1 Dynamic algorithms from static via lazy recomputation

The following observation allows one to obtain dynamic algorithms by using known static algorithms as subroutines.

► **Observation 2.** Given a  $t$ -respecting (resp.,  $t$ -approximation) static algorithm  $\mathcal{A}_S$  for MAX-CUT which runs in time  $T(m, n)$ , there exists a fully dynamic algorithm  $\mathcal{A}_D$  which maintains a  $(t - \epsilon)$ -respecting (resp.,  $(t - \epsilon)$ -approximate) cut for any constant  $\epsilon > 0$  in  $O(\frac{T(m, n)}{\epsilon m})$  worst-case update time.

The proof of Observation 2 is deferred to the Appendix. Using observation 2 gives the following fully dynamic algorithms. For any constant  $\epsilon > 0$ , a  $(\frac{1}{2} - \epsilon)$ -respecting cut can be maintained in  $O(1/\epsilon)$  worst-case update time by using Greedy Max-Cut. Similarly, the algorithm of [2] yields a dynamic algorithm to maintain a  $(0.878 - \epsilon)$ -approximate cut (w.h.p) for a fixed constant  $\epsilon > 0$  in  $O(\text{polylog}(n))$  worst case update time. For instances where the size of the cut-set of the optimal cut contains  $(\frac{1}{2} + \epsilon)|E|$  edges, the rounding algorithm of [13] can be used together with the algorithm of [2] to get a  $(\frac{1}{2} + \Omega(\frac{\epsilon}{\log 1/\epsilon}))$ -respecting cut in  $O(\text{polylog}(n))$  worst case update time where  $\epsilon > 0$  is a constant. However, to maintain a  $\frac{1}{2}$ -respecting cut for graphs in which the optimal cut is  $(\frac{1}{2} + O(\frac{1}{n}))$  the update time using this technique can be  $\tilde{\Omega}(n)$  time which is prohibitive. Thus there remains a need to design dynamic algorithms to maintain a  $\frac{1}{2}$ -respecting cut *exactly* in *sublinear* update time.

## 1.3 Our Contribution

We present the first fully dynamic algorithms in the sequential and distributed settings which *exactly* maintain a  $\frac{1}{2}$ -respecting cut. Our results are summarized in the following theorems.

► **Theorem 3.** *There exists a deterministic fully dynamic sequential algorithm which maintains a  $\frac{1}{2}$ -respecting cut, requires no more than one adjustment per update and takes  $O(\Delta)$  worst case update time, where  $\Delta$  denotes the maximum degree of the graph after the update.*

The algorithm in Theorem 3 is used as a subroutine in all other algorithms in this paper. The next result gives the first fully dynamic *deterministic* algorithm in the MPC setting with  $n$  machines and  $\Theta(n)$  memory per machine to maintain a  $\frac{1}{2}$ -respecting cut. Our algorithm takes  $O(1)$  rounds, requires no more than one adjustment and uses  $O(\Delta)$  total communication per round. This significantly improves on the parallel implementation of the static algorithm to maintain a  $\frac{1}{2}$ -respecting cut based on the idea of pairwise independence which can take as much as  $O(n^2 \log n)$  total communication and  $\Omega(n)$  adjustments.

► **Theorem 4.** *Given a graph on  $n$  vertices and  $m$  edges, there exists a deterministic fully dynamic MPC algorithm on  $n$  machines having  $\Theta(n)$  memory each, which maintains a  $\frac{1}{2}$ -respecting cut on  $G$  and takes  $O(1)$  rounds, makes at most one adjustment, and uses  $O(\Delta)$  bits of communication per update. If we start with an arbitrary graph, the preprocessing for the algorithm takes  $O(1)$  rounds and  $O(n^2 \log n)$  bits of communication.*

We note that the worst-case update time of  $O(\Delta)$  can be quite large in the case when  $\Delta = \Omega(n)$  and thus costly in the dynamic setting. This motivates the design of sublinear update time algorithms for all regimes of  $\Delta$ . Our next result is a sublinear (in  $m$ ) amortized update time algorithm which is useful for sufficiently sparse graphs having high maximum degree.

► **Theorem 5.** *There exists a deterministic fully dynamic sequential algorithm which maintains a  $\frac{1}{2}$ -respecting cut, and takes  $O(m^{1/2})$  amortized update time where  $m$  is the maximum number of edges in the graph during any arbitrary sequence of updates.*

Our final result is a randomized algorithm which always maintains a  $\frac{1}{2}$ -respecting cut and takes sublinear in  $n$  worst-case update time when updates come from an oblivious adversary.

► **Theorem 6.** *There exists a randomized fully dynamic sequential algorithm which maintains a  $\frac{1}{2}$ -respecting cut and takes  $\tilde{O}(n^{2/3})$  worst-case update time with high probability.*

We note that for our algorithms in Theorems 5 and 6, the adjustment cost can be  $\Omega(n)$ .

## 1.4 Our techniques

Our techniques utilize combinatorial and structural properties of cuts in graphs. The key insight underlying our algorithms is the following: in any cut  $C$  which is not  $\frac{1}{2}$ -respecting, there exists a vertex which can be moved across the cut to increase the size of  $C$ 's cut-set. We show that this vertex can be efficiently found, yielding a simple deterministic  $O(\Delta)$  worst case update time algorithm. This algorithm is not “local” in the sense that endpoints of the updated edge need not qualify as vertices which can be moved to increase the number of cut edges (Theorem 9). Such locality is often exploited to obtain dynamic and distributed algorithms for problems such as vertex cover, independent set and coloring. Despite this, we show that the algorithm can be used to get a deterministic fully dynamic distributed algorithm taking  $O(1)$  rounds and no more than one adjustment.

Central to our sublinear time algorithms of Theorem 5 and 6 is a *cut-combining* technique. This allows us to work on induced subgraphs of  $G$  and combine their “locally maintained” cuts to yield a  $\frac{1}{2}$ -respecting cut on  $G$ . However, the update time depends the complexity of maintaining  $\frac{1}{2}$ -respecting cuts on individual subgraphs and the combining step. We work around this non-trivial dependence. For our algorithm of Theorem 5, we partition vertices based on their degree and only *selectively* update data structures. We show that selective updating is sufficient for our purpose and refine the vertex partition after sufficiently many updates. This leads to a simple  $O(m^{1/2})$  amortized update time algorithm. To obtain the algorithm of Theorem 6, we extend the cut-combining idea and apply it to a random multi-way  $k$ -partition of  $V$  and obtain a sublinear in  $n$  worst case update time algorithm.

## 1.5 Organization of the paper

In the next section, we present an  $O(\Delta)$  update time algorithm. In Section 3, we present the dynamic distributed algorithm of Theorem 4. In Section 4, we give the  $O(m^{1/2})$  amortized update time sequential algorithm of Theorem 5. In section 5, we give the randomized algorithm of Theorem 6.

## 2 Preliminaries

Starting with an empty graph  $G = (V, E)$  where  $V = \{v_1, \dots, v_n\}$  is fixed, an update to  $G$  is either an insertion or a deletion of an edge  $\{v_i, v_j\}$  from  $E$ . For a cut  $C = (S, \bar{S})$  let  $\alpha_C(G) = \frac{|E(S, \bar{S})|}{|E|}$  denote the ratio of the sizes of  $C$ 's cut-set and  $E$ . The sizes of sets  $S, \bar{S}$  and the cut-set  $E(S, \bar{S})$  corresponding to the cut  $C = (S, \bar{S})$  are maintained by all algorithms to facilitate queries in constant time. Let  $G_k = (V, E_k)$  be the resulting graph after  $k$  updates have been made to  $G := G_0$  and  $m$  denote the number of edges in the graph at any given time. The degree of any vertex  $v$  in  $G_k$  is denoted by  $deg_k(v)$ .

The cut on  $G_0$ , the empty graph is initialized to  $(V, \emptyset)$ . Given a  $\frac{1}{2}$ -respecting cut  $C = (S, \bar{S})$ , i.e.  $\alpha_C(G_{k-1}) \geq \frac{1}{2}$  for some  $k \geq 1$ , there are a few cases to consider when an edge update  $\{v_i, v_j\}$  is made to  $G_{k-1}$ . Deletion of a non-cut edge or insertion of a cut edge never decreases the size of  $C$ 's cut-set. However,  $C$  needs to be updated if a cut edge is deleted, or a non-cut edge is inserted since  $C$  may cease to be  $\frac{1}{2}$ -respecting.

### 2.1 A crucial observation

We say that a vertex  $u$  is switched (with respect to a cut  $C = (S, \bar{S})$ ) if  $u$  is in  $S$  (resp.  $\bar{S}$ ) and moved to  $\bar{S}$  (resp.  $S$ ). We leverage the existence of vertices which can be switched to increase the size of the cut-set  $|E(S, \bar{S})|$  of  $C$  for *any* cut  $C$  which is not  $\frac{1}{2}$ -respecting. Thus, if  $C$  ceases to be  $\frac{1}{2}$ -respecting following any update there exists a vertex which can be switched to restore the  $\frac{1}{2}$ -respecting property.

► **Definition 7 (Switching vertex).** For a cut  $C = (S, \bar{S})$ , let  $N_S(u) = \{v \in S \mid (u, v) \in E\}$  be the neighbors of  $u$  in  $S$  and  $N_{\bar{S}}(u) = \{v \in \bar{S} \mid (u, v) \in E\}$  be the neighbors of  $u$  in  $\bar{S}$ . Then  $u$  is a switching vertex if one of the following two conditions holds: i)  $u \in S$  and  $|N_S(u)| - |N_{\bar{S}}(u)| \geq 1$  and ii)  $u \in \bar{S}$  and  $|N_{\bar{S}}(u)| - |N_S(u)| \geq 1$ .

► **Theorem 8.** Let  $C$  be a  $\frac{1}{2}$ -respecting cut w.r.t.  $G_{k-1}$  i.e.,  $\alpha_C(G_{k-1}) \geq \frac{1}{2}$  and  $\{v_i, v_j\}$  be an update. If  $\alpha_C(G_k) < \frac{1}{2}$ , then there exists a switching vertex  $u$  w.r.t.  $C$  such that if  $u$  is switched, then  $\alpha_C(G_k) \geq \frac{1}{2}$ .

**Proof.** Suppose there does not exist a switching vertex. Then,

$$\alpha_C(G_k) = \frac{1}{2|E_k|} \sum_{u \in V} \max\{|N_S(u)|, |N_{\bar{S}}(u)|\} \geq \frac{1}{2|E_k|} \sum_{v \in V} \frac{1}{2} deg_k(v) = \frac{1}{4|E_k|} 2|E_k| = \frac{1}{2}.$$

clearly contradicting our assumption that  $\alpha_C(G_k) < \frac{1}{2}$ . If a switching vertex  $u$  is switched, then the size of  $C$ 's cut set increases by at least 1 so that  $\alpha_C(G_k) \geq \frac{1}{2}$ . ◀

Given the count of a vertex's neighbors in  $S$  and  $\bar{S}$ , it can be decided whether it is switching or not. Maintaining these neighbor counts is necessary to determine a vertex to switch. However, testing all vertices whether they are switching is costly. In the next section we show how to efficiently maintain a set of switching vertices. The following theorem rules out the possibility of using end points of the updated edge as switching vertices. A proof can be found in the Appendix.

► **Theorem 9.** Given an edge update  $\{v_i, v_j\}$  to  $G_{k-1}$  for  $k \geq 1$ , and a  $\frac{1}{2}$ -respecting cut  $C_{k-1}$  maintained on  $G_{k-1}$ , a switching vertex with respect to  $C_{k-1}$  need not always be one of  $v_i, v_j$ .

## 2.2 An $O(\Delta)$ worst-case update time algorithm

In this section, we give a simple fully dynamic algorithm with worst case update time  $O(\Delta)$ .

**Data Structures.** For each vertex  $u \in V$  and a cut  $C = (S, \bar{S})$ , we maintain the following: i)  $N_S(u)$ : a list of neighbors of  $u$  in  $S$ , and its size  $|N_S(u)|$ , ii)  $N_{\bar{S}}(u)$ : a list of neighbors of  $u$  in  $\bar{S}$  and its size  $|N_{\bar{S}}(u)|$  and, iii)  $flag(u)$ : a bit which is 1 if  $u \in S$  and -1 if  $u \in \bar{S}$ .

► **Definition 10 (Gain of a vertex).** *The gain of a vertex  $u$  with respect to a cut  $C = (S, \bar{S})$  and denoted by  $\mathcal{G}(u)$  is given by  $\mathcal{G}(u) = flag(u)(|N_S(u)| - |N_{\bar{S}}(u)|)$ .*

The gain of a vertex  $u$  measures the change in the number of cut edges of  $C$ , if  $u$  is switched. Note that a vertex is switching if the gain is positive, and non-switching otherwise. The following (global) data structures are also maintained:

- a. A doubly linked list  $\mathcal{L}$ , which stores nodes corresponding to switching vertices.
- b. An array  $P$  where  $P[i]$  stores the gain of  $v_i$  and a pointer. The pointer points to the node in  $\mathcal{L}$  corresponding to  $v_i$  if  $\mathcal{G}(v_i) > 0$  and is *NULL* otherwise.

The head of  $\mathcal{L}$ , denoted by  $\mathcal{L}.head$  is *NULL* if no switching vertex exists. Each node of  $\mathcal{L}$  corresponding to a switching vertex  $v_i$  stores  $i$  as its value.

**Algorithm.** The algorithm begins with  $G_0$ , the empty graph and  $C = (S, \bar{S}) = (V, \emptyset)$  on  $G_0$ . It maintains a  $\frac{1}{2}$ -respecting cut on  $G_{k-1}$  for any  $k \geq 1$  as follows: when an edge update  $\{v_i, v_j\}$  to  $G_{k-1}$  arrives,  $N_S(v_i), N_{\bar{S}}(v_i), N_S(v_j), N_{\bar{S}}(v_j)$  are updated (including their sizes) along with  $P[i]$  and  $P[j]$ . If either of  $v_i, v_j$  become switching or non-switching,  $\mathcal{L}$  is appropriately modified.  $C$  is checked if it is  $\frac{1}{2}$ -respecting. If  $C$  ceases to be  $\frac{1}{2}$ -respecting then a switching vertex  $v_s$  is found by accessing the node pointed to by  $\mathcal{L}.head$  which stores the value  $s$ . This node is removed from  $\mathcal{L}$ ,  $v_s$  is switched and  $P[s]$  is updated. Data structures of  $v_t$  and  $P[t]$  of all neighbors  $v_t$  of  $v_s$  are modified to reflect  $v_s$ 's switch. Thereafter, depending on whether or not  $\mathcal{G}(v_t) > 0$  in the updated cut, the node corresponding to  $v_t$  in  $\mathcal{L}$  is inserted or removed. The pseudo code of the algorithm is as follows.

■ **Algorithm 1** Delta-Dynamic Max-Cut( $G_{k-1}, \{v_i, v_j\}, C = (S, \bar{S})$ ).

- 
- 1: Update  $N_S(v_i), N_S(v_j), N_{\bar{S}}(v_i), N_{\bar{S}}(v_j), \alpha_C(G_k), P[i], P[j]$ .
  - 2: **for**  $v_t \in \{v_i, v_j\}$  **do**
  - 3:     Add(remove) the node corresponding to  $v_t$  in  $\mathcal{L}$  if  $v_t$  becomes switching(non-switching).
  - 4: **end for**
  - 5: **if**  $\alpha_C(G_k) < \frac{1}{2}$  **then**
  - 6:      $v_s \leftarrow \mathcal{L}.head$ . Remove  $v_s$  from  $\mathcal{L}$ .
  - 7:     Switch  $v_s$  and update  $C, flag(v_s), N_S(v_s), N_{\bar{S}}(v_s), P[s]$ .
  - 8:     **for**  $v_t \in N_S(v_s) \cup N_{\bar{S}}(v_s)$  **do**
  - 9:         Update  $N_S(v_t)$  and  $N_{\bar{S}}(v_t)$  as appropriate.
  - 10:         Add(remove) the node corresponding to  $v_t$  in  $\mathcal{L}$  if  $v_t$  becomes switching(non-switching) and update  $P[t]$ .
  - 11:     **end for**
  - 12: **end if**
  - 13: **return**  $v_s$ .
-

**Running Time.** Updates to data structures of  $v_i, v_j$  and  $P[i], P[j]$  take constant time. Inserting or removing a node from  $\mathcal{L}$  also takes constant time. Switching  $v_s$  in the case when  $C$  is no longer  $\frac{1}{2}$ -respecting takes time proportional to updating all its neighbors' data structures, their corresponding entries in  $P$  and their corresponding nodes in  $\mathcal{L}$ . This takes  $O(\Delta)$  time. Theorem 3 follows.

### 3 A fully dynamic distributed algorithm

In this section, we present the algorithm of Theorem 4. Dynamic distributed algorithms have been well studied in the past [36, 16], and techniques to design sequential fully dynamic algorithms are often applicable in designing their distributed counterparts. As an example, for the maximal independent set problem the distributed implementation of the dynamic sequential algorithm of Assadi et al. [3] improves on the dynamic distributed algorithm of Censor-Hillel et al. [10]. This is often easier for problems in which only the neighborhood of vertices incident to an update needs to be examined to restore the maintained property. For MAX-CUT, it may not always be the case that endpoints of the update edge can be switched to maintain a  $\frac{1}{2}$ -approximate cut by Theorem 9. Nevertheless, we show how to use the  $O(\Delta)$  update time algorithm to get an efficient fully dynamic distributed algorithm in the MPC model.

In the model we consider, there are  $n$  machines  $M_1, \dots, M_n$  each corresponding to vertices  $v_1, \dots, v_n$  respectively. Given a graph  $G = (V, E)$  where  $n = |V|$  and  $m = |E|$ , each machine  $M_i$  initially stores a list of neighbors of  $v_i$  in addition to storing  $f(v)$  and  $R \subseteq [k]$  to run the static distributed algorithm obtained by pairwise independence (see Section 1.1) and obtain an initial  $\frac{1}{2}$ -respecting cut  $C = (S, \bar{S})$  on  $G$ . For any  $i, j \in [n]$  we say that machine  $M_i$  is a neighbor of  $M_j$  if  $(v_i, v_j) \in E$ . We let  $M_n$  be the coordinator machine which stores the position of any vertex  $v \in V$  in  $C$ , i.e. whether  $v \in S$  or  $\bar{S}$ . Given the initial cut  $C$ , each machine  $M_i$  maintains whether  $v_i$  is a switching vertex w.r.t.  $C$  or not. This can be done in a single round and  $O(m)$  total communication—every machine simply sends the position of its corresponding vertex in  $C$  to all its neighbors. We also ensure that the coordinator  $M_n$  maintains the list of all switching vertices w.r.t.  $C$ , the total number of edges in the graph and the size of  $C$ 's cut-set. After this initial preprocessing which takes  $O(1)$  rounds and  $O(n^2 \log n)$  bits of communication, the information  $f(v)$  and  $R$  stored by all machines can be discarded.

We now describe the update algorithm. Whenever an update  $\{v_i, v_j\}$  is made to  $G$ , machines  $M_i$  and  $M_j$  are informed and thereafter, they update their list of neighbors. Both  $M_i$  and  $M_j$  inform the coordinator  $M_n$  of the update in addition to informing whether  $v_i$  and  $v_j$  become switching w.r.t. the maintained cut  $C$ . This allows  $M_n$  to update  $m$ , size of the cut-set  $C$  and the set of switching vertices. If  $C$  ceases to be  $\frac{1}{2}$ -respecting,  $M_n$  selects an arbitrary switching vertex,  $v_s$  and informs  $M_s$ . Thereafter,  $M_s$  updates its local data structures to reflect the switch and informs all its neighbors to reflect the switch. If any neighbor  $v_k$  of  $v_s$  becomes a switching vertex w.r.t. the updated cut  $C$ ,  $M_k$  informs the coordinator  $M_n$ , after which  $M_n$  updates the list of switching vertices.

This takes  $O(1)$  rounds,  $O(\Delta)$  total communication per round and at most one adjustment to  $C$  after any edge update. Theorem 4 follows.

#### 4 Achieving sublinear (in $m$ ) update time

In this section, we present an  $O(m^{1/2})$  amortized update algorithm which improves on the  $O(\Delta)$  update time algorithm for sufficiently sparse graphs having high maximum degree. The high level ideas involve: i) partitioning the graph  $G$  into induced subgraphs  $G_1$  and  $G_2$  on  $V_{low}$  and  $V_{high}$  respectively where  $V_{low}$  and  $V_{high}$  are sets of low and high degree vertices respectively, ii) combining  $\frac{1}{2}$ -respecting cuts  $C_1$  and  $C_2$  on  $G_1$  and  $G_2$  respectively which are maintained using the algorithm of Theorem 3 and, iii) *selectively* updating data structures. The latter idea is crucial to reduce the update time. When a high degree vertex  $v \in V_{high}$  switches w.r.t. the cut  $C_2$ , data structures of only its neighbors in  $V_{high}$  are updated leading to stale information in data structures of its neighbors in  $V_{low}$ . A similar idea was used in the fully dynamic algorithm for the maximal independent set problem [3]. We show that lazy updating of low degree vertex data structures is sufficient for our purpose and re-build  $G_1$  and  $G_2$  after sufficiently many updates which leads to  $O(m^{1/2})$  amortized update time. Given  $\frac{1}{2}$ -respecting cuts on any vertex disjoint induced subgraphs of  $G$ , we first show that they can be combined to give a  $\frac{1}{2}$ -respecting on  $G$ .

► **Theorem 11** (Cut combining). *Let  $G = (V, E)$  be any graph and  $C_1 = (S, \bar{S})$  and  $C_2 = (T, \bar{T})$  be  $\frac{1}{2}$ -respecting cuts with respect to the vertex disjoint induced subgraphs  $G_1 = (V_1, E_1)$ ,  $G_2 = (V_2, E_2)$  of  $G$  such that  $S \cup \bar{S} = V_1$ ,  $T \cup \bar{T} = V_2$  and  $V_1 \cup V_2 = V$ . Then one of the following is a  $\frac{1}{2}$ -respecting cut  $C$  of  $G$ :*

- i)  $(S \cup T, \bar{S} \cup \bar{T})$
- ii)  $(S \cup \bar{T}, \bar{S} \cup T)$ .

A formal proof of Theorem 11 is omitted for the sake of brevity but it follows by noting that cut-edges of  $C_1$  and  $C_2$  remain cut edges in both cuts considered in i) and ii), and the cut-set of one of the cuts in i) and ii) must contain half of the remaining edges.

**Data Structures.** For any  $U, W \subseteq V$ , let  $E(U, W)$  be the set of edges having one endpoint in  $U$  and the other in  $W$ . To determine  $C$ , the following edge counts are maintained:  $|E(S, T)|$ ,  $|E(S, \bar{T})|$ ,  $|E(\bar{S}, T)|$ ,  $|E(\bar{S}, \bar{T})|$ . If  $|E(S, \bar{T})| + |E(\bar{S}, T)| \geq |E(S, T)| + |E(\bar{S}, \bar{T})|$ , then  $C = (S \cup T, \bar{S} \cup \bar{T})$ , else we take  $C = (S \cup \bar{T}, \bar{S} \cup T)$ . Let  $N_U(v)$  denote the list of neighbors of  $v$  in  $U \subseteq V$ . In addition to data structures required by the algorithm of Theorem 3, every vertex  $v \in V_{low}$  maintains neighbor counts  $N_T(v)$ ,  $N_{\bar{T}}(v)$  and every vertex  $v \in V_{high}$  maintains neighbor counts  $N_S(v)$ ,  $N_{\bar{S}}(v)$ . For any subset  $U, W \subseteq V$  s.t.  $U \in \{S, \bar{S}\}$  and  $W \in \{T, \bar{T}\}$ , note that the edge count  $|E(U, W)| = \sum_{u \in U} |N_W(u)|$ .

The main challenge is to correctly maintain these edge counts without updating all the neighbors of a high degree vertex which switches w.r.t  $C_2$ . These edge counts change if i) an edge update  $(v_i, v_j)$  is encountered and/or ii) a vertex switches w.r.t. either  $C_1$  or  $C_2$ . Our update algorithm switches at most a single vertex w.r.t  $C_1$  or  $C_2$  and maintains neighbor counts of high degree vertices accurately at any given time. Combined with recomputing neighbor counts of low degree vertices *only* when they switch, this is sufficient to maintain edge counts correctly at any given time.

**Algorithm.** The algorithm consists of phases. The  $k^{th}$  phase for  $k \geq 1$  begins with the graph  $G$  containing  $m_k$  edges and  $\frac{1}{2}$ -respecting cuts  $C_1$  and  $C_2$  on the induced subgraphs  $G_1$  and  $G_2$  respectively. Here,  $G_1$  and  $G_2$  are induced subgraphs on  $V_{low} = \{v \in V | deg(v) \leq m_k^{1/2}\}$  and  $V_{high} = V \setminus V_{low}$  respectively. We assume that the first phase starts with a single edge, i.e.  $m_1 = 1$ . The  $k^{th}$  phase consists of  $m_k^{1/2}$  updates after which a new phase corresponding to

the new value of  $m_k$  begins. Thereafter, all data structures are reinitialized and  $\frac{1}{2}$ -respecting cuts are computed for  $G_1$  and  $G_2$  (under the new value of  $m_k$ ). The total time taken to reinitialize a phase is  $O(m_k)$ , leading to  $O(m_k^{1/2})$  amortized update time.

Note that the number of high degree vertices for any phase beginning with  $m_k$  edges is bounded by  $|V_{high}| = O(m_k)/\Omega(m_k^{1/2}) = O(m_k^{1/2})$ . Let  $\{v_i, v_j\}$  be an edge update during the  $k^{th}$  phase for  $k \geq 1$ . Then,

1. if  $v_i \in V_{low}, v_j \in V_{high}$ : One of the lists  $N_T(v_i), N_{\bar{T}}(v_i)$  and one of  $N_{\bar{S}}(v_j), N_S(v_j)$  is updated. Additionally, one of the edge counts  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$  depending on the position of  $v_i$  and  $v_j$  in  $C_1$  and  $C_2$  respectively, is updated.
2. if  $v_i, v_j \in V_{low}$ : the algorithm of Theorem 3 is used to restore  $C_1$ . Let  $u$  be a vertex which is switched w.r.t  $C_1$ . All data structures of high degree neighbors of  $u \in N_T(u) \cup N_{\bar{T}}(u)$  are updated. Moreover,  $u$  recomputes the lists of its high degree neighbors  $N_T(u), N_{\bar{T}}(u)$ . The edge counts  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$  are updated.
3. if  $v_i, v_j \in V_{high}$ : the algorithm of Theorem 3 is used to restore  $C_2$ . The edge counts  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$  are updated.

The pseudo code of the update algorithm is as follows.

■ **Algorithm 2** Sublinear Max-Cut ( $\{v_i, v_j\}, C_1 = (S, \bar{S}), C_2 = (T, \bar{T})$ ).

---

```

1: if  $v_i \in V_{low}$  and  $v_j \in V_{high}$  then
2:   Update  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|, N_T(v_i), N_{\bar{T}}(v_i), N_S(v_j), N_{\bar{S}}(v_j)$ .
3: else
4:   if  $v_i, v_j \in V_{low}$  then
5:      $u \leftarrow$  Delta-Dynamic Max-Cut( $G_1, \{v_i, v_j\}, C_1$ ).
6:     for  $w \in N_T(u) \cup N_{\bar{T}}(u)$  do
7:       Update  $N_S(w), N_{\bar{S}}(w)$  to reflect the new position of  $u$  in the cut  $(S, \bar{S})$ .
8:     end for
9:     Update  $N_T(u), N_{\bar{T}}(u), |E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$ .
10:   end if
11:   if  $v_i, v_j \in V_{high}$  then
12:      $u \leftarrow$  Delta-Dynamic Max-Cut( $G_2, \{v_i, v_j\}, C_2$ ).
13:     Update  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$ .
14:   end if
15: end if

```

---

**Running Time.** If an update  $\{v_i, v_j\}$  is such that  $v_i \in V_{low}, v_j \in V_{high}$ , the update time is  $O(1)$ .

If  $v_i, v_j \in V_{low}$  the call to the  $O(\Delta)$  update time algorithm takes time  $O(m_k^{1/2})$  since any vertex in  $V_{low}$  has degree at most  $2m_k^{1/2} = O(m_k^{1/2})$  throughout the phase, by definition. Updating the list of neighbors of the switched vertex  $u$ , and updating the data structures of  $u$ 's neighbors takes  $O(m_k^{1/2})$  time. Updating edge counts takes constant time since they are incremented or decremented by constants which can be determined from the size of neighbor lists of  $u$ .

If  $v_i, v_j \in V_{high}$ : the call to the  $O(\Delta)$  update time algorithm takes time  $O(m_k^{1/2})$  since  $|V_{high}| = O(m_k^{1/2})$ . As in the second case, updating edge counts takes constant time.

Thus, the time taken to handle an edge update during a phase beginning with  $m_k$  edges is  $O(m_k^{1/2})$ . Since the amortized cost of re-initialization is  $O(m_k^{1/2})$ , this gives an  $O(m^{1/2})$  amortized update time algorithm where  $m$  denotes the maximum number of edges in  $G$  during an arbitrary sequence sequence of updates. Theorem 5 follows. A proof of correctness can be found in the Appendix.

## 5 Achieving sublinear (in $n$ ) worst case update time

In this section we give a randomized algorithm which exactly maintains a  $\frac{1}{2}$ -respecting cut and takes  $\tilde{O}(n^{2/3})$  worst case update time w.h.p. We obtain the result by first designing an algorithm with  $O(n^{2/3})$  *expected* worst-case update time. Then, we apply the probability amplification result in [7] which gives a  $\tilde{O}(n^{2/3})$  worst-case update time algorithm w.h.p.

The high level idea of our algorithm is to use cut-combining idea on  $k$  vertex disjoint subgraphs  $G_1, G_2, \dots, G_k$  induced by a random  $k$ -partition of  $V$  denoted by  $(V_1, V_2, \dots, V_k)$ . The random partition is constructed using the oracle described in Section 1.2 such that  $\bigcup_{i=1}^k V_i = V$  and  $|V_1| = |V_2| = \dots = |V_{k-1}| = \lceil n/k \rceil$ ,  $|V_k| = n - (k-1)\lceil n/k \rceil$ . On each subgraph  $G_i$  induced by  $V_i$ , a  $\frac{1}{2}$  respecting cut  $C_i = (S_i, \bar{S}_i)$  (where  $\bar{S}_i = V_i \setminus S_i$ ) is dynamically maintained using the algorithm of Theorem 3. We now describe the data structures and the update algorithm.

**Data structures.** In addition to data structures required by the  $O(\Delta)$ -update time algorithm, we maintain: i) For each vertex  $v \in V$ , lists of its neighbors in each  $S_i$ , (denoted by  $N_{S_i}(v)$ ) and  $\bar{S}_i$  (denoted by  $N_{\bar{S}_i}(v)$ ) for all  $1 \leq i \leq k$  and, ii) For all  $1 \leq i, j \leq k$ , the edge counts  $|E(S_i, S_j)|$ ,  $|E(S_i, \bar{S}_j)|$ ,  $|E(\bar{S}_i, S_j)|$ ,  $|E(\bar{S}_i, \bar{S}_j)|$  for a total of  $\binom{2k}{2} = O(k^2)$  counts. The edge counts can be maintained using the size of neighbor lists maintained for each vertex.

### Algorithm.

Cut combining: We first describe how to combine  $\frac{1}{2}$ -approximate cuts  $C_i$  on  $G_i$  for  $1 \leq i \leq k$  to get a  $\frac{1}{2}$ -approximate cut  $C$ , on  $G$ . Initially,  $C = (S_1, \bar{S}_1)$ . Whenever considering cut  $C_i = (S_i, \bar{S}_i)$  for  $2 \leq i \leq k$  to combine with  $C$ , the edge counts  $|E(S_i, S_j)|$ ,  $|E(S_i, \bar{S}_j)|$ ,  $|E(\bar{S}_i, S_j)|$ ,  $|E(\bar{S}_i, \bar{S}_j)|$ , for  $1 \leq j \leq i-1$  are used to compute the edge counts  $|E(S, S_i)|$ ,  $|E(S, \bar{S}_i)|$ ,  $|E(\bar{S}, S_i)|$ ,  $|E(\bar{S}, \bar{S}_i)|$ . Depending on the combination which maximizes  $|E(S, \bar{S})|$ , either  $S_i$  (resp.  $\bar{S}_i$ ) is added to  $S$  (resp.  $\bar{S}$ ) or  $S_i$  (resp.  $\bar{S}_i$ ) is added to  $\bar{S}$  (resp.  $S$ ). Computing the edge counts takes  $O(k)$  time, yielding  $O(k^2)$  time to compute  $C$ .

Update algorithm: Let  $\{v_i, v_j\}$  be an edge update. Then,

1. if  $v_i \in V_p$  and  $v_j \in V_q$  s.t.  $p \neq q$ : Only the lists  $N_{S_q}(v_i)$ ,  $N_{\bar{S}_q}(v_i)$ ,  $N_{S_p}(v_j)$ ,  $N_{\bar{S}_p}(v_j)$  and edge counts  $|E(S_p, S_q)|$ ,  $|E(S_p, \bar{S}_q)|$ ,  $|E(\bar{S}_p, S_q)|$ ,  $|E(\bar{S}_p, \bar{S}_q)|$  are updated which takes  $O(1)$  time.
2. if  $v_i, v_j \in V_p$  for some  $p$ : the cut  $C_p$  is updated using the  $O(\Delta)$  update time algorithm. Let  $u$  be the switched vertex w.r.t  $C_p$ . The lists  $N_{S_p}(w)$ ,  $N_{\bar{S}_p}(w)$  of all neighbors  $w$  of  $u$  are updated to reflect  $u$ 's switch. For all  $1 \leq q \leq k$  such that  $N_{S_q}(u) \cup N_{\bar{S}_q}(u) \neq \emptyset$ , edge counts of the form  $|E(S_p, S_q)|$ ,  $|E(S_p, \bar{S}_q)|$ ,  $|E(\bar{S}_p, S_q)|$ ,  $|E(\bar{S}_p, \bar{S}_q)|$  are also updated. This can be done by using the values of  $|N_{S_q}(u)|$  and  $|N_{\bar{S}_q}(u)|$ .

Following this, the cuts  $C_1, \dots, C_k$  are combined to yield  $C$ . The pseudo code of the update algorithm is as follows.

Note that the only information required to determine how to combine the cut  $(S_t, \bar{S}_t)$  with  $(S, \bar{S})$  in each iteration of the for loop is the position of all  $S_i, \bar{S}_i$  for all  $i \leq t-1$  in  $(S, \bar{S})$ . Thus, computing the edge counts  $|E(S \cup S_t, \bar{S} \cup \bar{S}_t)|$ ,  $|E(S \cup \bar{S}_t, \bar{S} \cup S_t)|$  can be done in  $O(k)$  time, and lines 14 and 16 of Algorithm 3 do not need to be explicitly implemented.

**Running Time.** For the case when  $v_i \in V_p$  and  $v_j \in V_q$  s.t.  $p \neq q$  updating the edge counts takes constant time. However, the combining cost is incurred. This is because a single update can possibly cause the cuts to combine differently in order to maintain a  $\frac{1}{2}$ -respecting cut on  $G$ .

■ **Algorithm 3** Randomized Sublinear MAX-CUT ( $\{v_i, v_j\}, G_1, \dots, G_k, C_1, \dots, C_k$ ).

---

```

1: if  $v_i \in V_p, v_j \in V_q$  s.t.  $p \neq q$  then
2:   Update  $N_{S_q}(v_i), N_{\bar{S}_q}(v_i), N_{S_p}(v_j), N_{\bar{S}_p}(v_j)$ .
3:   Update  $|E(S_p, S_q)|, |E(\bar{S}_p, S_q)|, |E(S_p, \bar{S}_q)|, |E(\bar{S}_p, \bar{S}_q)|$  appropriately.
4: else
5:    $u \leftarrow$  Delta-Dynamic Max-Cut( $G_p, \{v_i, v_j\}, C_p$ ).
6:   for all neighbors  $v$  of  $u$  where  $v \in V_r$  for any  $1 \leq r \leq k$  do
7:     Update  $N_{S_r}(u), N_{\bar{S}_r}(u), N_{S_p}(v), N_{\bar{S}_p}(v)$ .
8:     Update  $|E(S_p, S_r)|, |E(\bar{S}_p, S_r)|, |E(S_p, \bar{S}_r)|, |E(\bar{S}_p, \bar{S}_r)|$  appropriately.
9:   end for
10: end if
11:  $S = S_1, \bar{S} = \bar{S}_1$ .
12: for  $t = 2, \dots, k$  do
13:   if  $|E(S \cup S_t, \bar{S} \cup \bar{S}_t)| \geq |E(S \cup \bar{S}_t, \bar{S} \cup S_t)|$  then
14:      $S = S \cup S_t, \bar{S} = \bar{S} \cup \bar{S}_t$ .
15:   else
16:      $S = S \cup \bar{S}_t, \bar{S} = \bar{S} \cup S_t$ .
17:   end if
18: end for

```

---

For the case when  $v_i, v_j \in V_p$  for some  $p$ , the algorithm of Theorem 3 takes  $O(n/k)$  time. Let  $u$  be the switched vertex w.r.t.  $C_p$ . Updating the neighbor lists of all neighbors of  $u$  takes  $O(\Delta)$  time. Thus, the update time in this case is  $O(\Delta + \frac{n}{k} + k^2) = O(\Delta + k^2)$ .

► **Lemma 12.** *The running time of the update algorithm is  $O(\frac{\Delta}{k} + k^2)$ . With  $k = \Theta(n^{1/3})$ , this yields  $O(n^{2/3})$  expected worst-case update time.*

**Proof.** Let  $\{v_i, v_j\}$  be an edge update. The probability that this update is of the second type, i.e.  $v_i, v_j \in V_p$  for some  $p \in [k]$  is at most  $1/k$ . The expected update time, denoted by  $E[T(n, k)]$  can be written as,

$$\begin{aligned}
E[T(n, k)] &= \Pr[v_i, v_j \in V_p]O(\Delta + k^2) + \Pr[v_i \in V_p, v_j \in V_q, p \neq q]O(k^2) \\
&= \Pr[v_i, v_j \in V_p]O(\Delta + k^2) + (1 - \Pr[v_i, v_j \in V_p])O(k^2) \\
&= \frac{1}{k}O(\Delta) + O(k^2) \\
&= O(\frac{\Delta}{k} + k^2) \\
&= O(\frac{n}{k} + k^2).
\end{aligned}$$

The value of  $k$  which minimizes  $E[T(n, k)]$  is  $\Theta(n^{1/3})$  yielding  $O(n^{2/3})$  expected worst case update time. ◀

Bernstein et al. [7] give a general technique to convert a fully dynamic data structure with expected worst-case update time to one with a worst-case update time with high probability. See [7] for technical details. By using their technique as a black-box, we convert our randomized algorithm described in this section taking  $O(n^{2/3})$  expected worst-case update time to one taking  $O(n^{2/3} \log^2(n)) = \tilde{O}(n^{2/3})$  update time with high probability. Theorem 6 follows.

## 6 Conclusion

The following open problems arise from our work. First, it would be interesting to improve on the algorithm in Theorem 5 to get a better update time in the *worst-case*. Second, the Algorithm in Theorem 6 works only for an oblivious adversary, and it would be interesting to design a randomized worst-case algorithm with better update time which works against an adaptive adversary.

We believe that ideas from our fully dynamic distributed MPC algorithm may be useful in other models such as the ones considered in [27, 39]. We observe that our dynamic algorithm for MPC can be implemented in the Congested-Clique model. Moreover, we believe that a dynamic MPC algorithm to maintain a  $\frac{1}{2}$ -respecting cut using only sublinear (in  $n$ ) memory per machine (in contrast to  $\Omega(n)$  memory as in the algorithm of Theorem 4) may be possible without a blow up in the round, adjustment or message complexity. A natural open question is whether there exists a deterministic fully dynamic algorithm with  $o(\Delta)$  round complexity and  $O(1)$  adjustment and message complexity to preserve a  $\frac{1}{2}$ -respecting cut in the *CONGEST* model. This may necessitate new techniques and lead to interesting connections to other fundamental problems studied in the distributed computing and dynamic algorithms literature.

---

## References

- 1 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 574–583, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591805.
- 2 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *J. ACM*, 63(2):12:1–12:35, May 2016. doi:10.1145/2837020.
- 3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 815–826, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188922.
- 4 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in  $n$  update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1919–1936, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=3310435.3310551>.
- 5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in  $o(\log n)$  update time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 383–392, October 2011. doi:10.1109/FOCS.2011.89.
- 6 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, page 273–284, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2463664.2465224.
- 7 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.
- 8 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In Artur Czumaj, editor, *Proceedings of the Twenty-*

- Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.
- 9 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Dynamic algorithms via the primal-dual method. *Inf. Comput.*, 261(Part):219–239, 2018. doi:10.1016/j.ic.2018.02.005.
  - 10 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, page 217–226, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2933057.2933083.
  - 11 Keren Censor-Hillel, Rina Levy, and Hadas Shachnai. Fast distributed approximation for max-cut. In Antonio Fernández Anta, Tomasz Jurdzinski, Miguel A. Mosteiro, and Yanyong Zhang, editors, *Algorithms for Sensor Systems - 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, volume 10718 of *Lecture Notes in Computer Science*, pages 41–56. Springer, 2017. doi:10.1007/978-3-319-72751-6\_4.
  - 12 K. C. Chang and D. H. . Du. Efficient algorithms for layer assignment problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(1):67–78, 1987.
  - 13 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, page 54–60, USA, 2004. IEEE Computer Society. doi:10.1109/FOCS.2004.39.
  - 14 Kwan-Wu Chin, Sieteng Soh, and Chen Meng. Novel scheduling algorithms for concurrent transmit/receive wireless mesh networks. *Computer Networks*, 56:1200–1214, March 2012. doi:10.1016/j.comnet.2011.12.001.
  - 15 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond, 2019. arXiv:1910.08025.
  - 16 Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Distributed computation in dynamic networks via random walks. *Theor. Comput. Sci.*, 581(C):45–66, May 2015. doi:10.1016/j.tcs.2015.02.044.
  - 17 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. doi:10.1145/1327452.1327492.
  - 18 Laxman Dhulipala, David Durfee, Janardhan Kulkarni, Richard Peng, Saurabh Sawlani, and Xiaorui Sun. Parallel batch-dynamic graphs: Algorithms and lower bounds. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1300–1319, USA, 2020. Society for Industrial and Applied Mathematics.
  - 19 Michel X. Goemans and David P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2sat. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 422–431, 1994. doi:10.1145/195058.195216.
  - 20 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22nd International Conference on Algorithms and Computation*, ISAAC'11, page 374–383, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-25591-5\_39.
  - 21 Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. A tree structure for dynamic facility location. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 39:1–39:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.39.
  - 22 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of*

- Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550. ACM, 2017. doi:10.1145/3055399.3055493.
- 23 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.
  - 24 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.
  - 25 Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, and David Saulpic. Dominating sets and connected dominating sets in dynamic graphs. In *STACS*, 2019.
  - 26 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48:723–760, July 2001. doi:10.1145/276698.276715.
  - 27 Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, page 49–58, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3323165.3323202.
  - 28 David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, December 1974. doi:10.1016/S0022-0000(74)80044-9.
  - 29 Satyen Kale and C. Seshadhri. Combinatorial approximation algorithms for maxcut using random walks. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 367–388. Tsinghua University Press, 2011. URL: <http://conference.iis.tsinghua.edu.cn/ICS2011/content/papers/20.html>.
  - 30 Bruce Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1131–1142, January 2013. doi:10.1137/1.9781611973105.81.
  - 31 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 938–948, USA, 2010. Society for Industrial and Applied Mathematics.
  - 32 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming*, 1972.
  - 33 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Deterministic worst case dynamic connectivity: Simpler and faster. *CoRR*, abs/1507.05944, 2015. arXiv:1507.05944.
  - 34 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 146–154, 2004. doi:10.1109/FOCS.2004.49.
  - 35 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, April 2007. doi:10.1137/S0097539705447372.
  - 36 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 513–522, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806760.
  - 37 M. Luby. Removing randomness in parallel computation without a processor penalty. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 162–173, 1988.
  - 38 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. doi:10.1145/2700206.
  - 39 Krzysztof Nowicki and Krzysztof Onak. Dynamic graph algorithms with batch updates in the massively parallel computation model, 2020. arXiv:2002.07800.

- 40 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010. doi:10.1145/1806689.1806753.
- 41 M. Preissmann and Andras Sebo. Optimal cuts in graphs and statistical mechanics. *Mathematical and Computer Modelling - MATH COMPUT MODELLING*, 26:1–11, October 1997. doi:10.1016/S0895-7177(97)00195-7.
- 42 Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, January 1981. doi:10.1145/322234.322235.
- 43 Shay Solomon. Fully dynamic maximal matching in constant update time. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.
- 44 José A. Soto. Improved analysis of a max-cut algorithm based on spectral partitioning. *SIAM J. Discret. Math.*, 29(1):259–268, 2015. doi:10.1137/14099098X.
- 45 Mikkel Thorup. Fully-dynamic min-cut. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 224–230, 2001. doi:10.1145/380752.380804.
- 46 L. Trevisan. Max cut and the smallest eigenvalue. *SIAM Journal on Computing*, 41(6):1769–1786, 2012. doi:10.1137/090773714.
- 47 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time, 2016. arXiv:1611.02864.
- 48 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud’10*, page 10, USA, 2010. USENIX Association.

## 7 Appendix

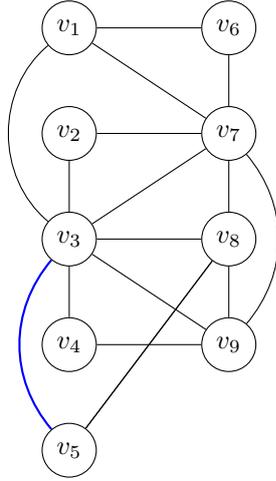
### 7.1 Proof of Observation 2

**Proof.** The high level idea is to partition the update sequence into phases consisting of  $O(\epsilon m)$  updates and spreading the time to recompute a  $t$ -respecting (resp.,  $t$ -approximate) cut using  $\mathcal{A}_S$  over any phase. Let  $P_i$  denote phase  $i$ ,  $G_{P_i}$  the graph at the beginning of phase  $i$  and  $m_i$  the number of edges in  $G_{P_i}$ . We let  $m_i = m$  so that phases  $P_{i+1}$  and  $P_{i+2}$  begin after  $\frac{\epsilon m}{2}$  and  $\epsilon m$  updates have been made to  $G_{P_i}$ , respectively. Algorithm  $\mathcal{A}_S$  is used to compute a  $t$ -respecting (resp.,  $t$ -approximate) cut  $C_{P_i}$  on  $G_{P_i}$  by spending  $T(m, n)$  time spread over updates between phase  $P_i$  and  $P_{i+1}$ , and  $C_{P_i}$  is used to answer all queries between phase  $P_{i+1}$  and  $P_{i+2}$ . This takes  $\frac{2T(m, n)}{\epsilon m} = O(\frac{T(m, n)}{\epsilon m})$  worst-case update time where  $C_{P_i}$  is a  $(t - \epsilon)$ -respecting (resp.,  $t$ -approximate) cut until phase  $P_{i+2}$  begins. Moreover, after  $P_{i+1}$  begins,  $\mathcal{A}_S$  is used to compute a  $t$ -respecting (resp.,  $t$ -approximate) cut  $C_{P_{i+1}}$  on  $G_{P_{i+1}}$  by spending  $T(m_{i+1}, n)$  time spread over updates between phase  $P_{i+1}$  and  $P_{i+2}$ , yielding a worst-case update time of  $\frac{2T(m_{i+1}, n)}{\epsilon m} \leq \frac{2T(m(1+\epsilon/2), n)}{\epsilon m} = O(\frac{T(m, n)}{\epsilon m})$ . Thus, the total worst-case update time is bounded by  $O(\frac{T(m, n)}{\epsilon m})$ . ◀

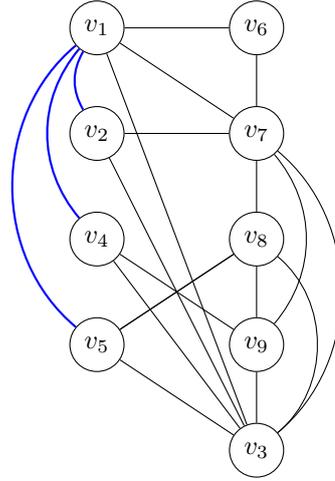
### 7.2 Endpoints of an updated edge may not be switching

▶ **Theorem 9.** *Given an edge update  $\{v_i, v_j\}$  to  $G_{k-1}$  for  $k \geq 1$ , and a  $\frac{1}{2}$ -respecting cut  $C_{k-1}$  maintained on  $G_{k-1}$ , a switching vertex with respect to  $C_{k-1}$  need not always be one of  $v_i, v_j$ .*

**Proof.** We refer to Figures 7.1 and 7.2 for the sake of illustration. Let  $V = \{v_1, \dots, v_9\}$  be the set of vertices such that  $S = V, \bar{S} = \emptyset$ . Consider the following sequence of edge insertions  $\{v_1, v_6\}, \{v_1, v_7\}, \{v_2, v_7\}, \{v_3, v_7\}, \{v_3, v_8\}, \{v_3, v_9\}, \{v_4, v_9\}, \{v_5, v_8\}$  which leads to



■ **Figure 7.1**  $S = \{v_1, \dots, v_5\}, \bar{S} = \{v_6, \dots, v_9\}$ . After  $\{v_3, v_5\}$  is added,  $v_3$  switches.



■ **Figure 7.2** After  $v_3$  switches and edges  $\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}$  are added, none of  $v_1, v_2, v_4, v_5$  are switching, yet the cut ceases to be  $\frac{1}{2}$  respecting.

$v_6, v_7, v_8, v_9$  moving to  $\bar{S}$  in that order, as a result. Next, consider the following non-cut edge insertions in no particular order:  $\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_6, v_7\}, \{v_7, v_8\}, \{v_8, v_9\}, \{v_7, v_9\}$ . The latter set of edge insertions does not make any vertex switching, After the edge  $\{v_3, v_5\}$  is added  $v_3$  switches to  $\bar{S}$ . Now consider the insertion of non-cut edges  $\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}$  so that none of their endpoints namely  $v_1, v_2, v_4, v_5$  become switching. But,  $(S, \bar{S})$  is no longer  $\frac{1}{2}$ -respecting. ◀

## 7.3 On the sublinear (in $m$ ) update time algorithm

### 7.3.1 Proof of correctness

► **Lemma 13.** *Algorithm 2 correctly maintains the edge counts  $|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|$  where  $C_1 = (S, \bar{S}), C_2 = (T, \bar{T})$ .*

**Proof.** Assume that the edge counts  $(|E(S, T)|, |E(S, \bar{T})|, |E(\bar{S}, T)|, |E(\bar{S}, \bar{T})|)$  are accurate before Algorithm 2 is executed to handle the edge update  $\{v_i, v_j\}$ . For  $v_i \in V_{low}$  and  $v_j \in V_{high}$  let  $X \in \{S, \bar{S}\}, Y \in \{T, \bar{T}\}$  be such that  $v_i \in X, v_j \in Y$ . If  $\{v_i, v_j\}$  is an edge insertion, then  $v_i$  is added to  $N_X(v_j)$ ,  $v_j$  to  $N_Y(v_i)$  and  $|E(X, Y)|$  is increased by 1. On the other hand, if  $\{v_i, v_j\}$  is an edge deletion,  $v_i$  is removed from  $N_X(v_j)$ ,  $v_j$  from  $N_Y(v_i)$  and  $|E(X, Y)|$  is decremented by 1. Thus, the edge counts are correctly updated in this case.

In the case when  $v_i, v_j \in V_{low}$ , Algorithm 1 is called in order to handle the edge update with respect to the induced subgraph  $G_1$ . Let  $u \in V_{low}$  be a switched vertex and let  $X, \bar{X} \in \{S, \bar{S}\}$  be such that  $u \in X$  moves to  $\bar{X}$  after the switch. Now,  $u$  may no longer have an accurate count of its neighbors in  $T$  and  $\bar{T}$  since when high degree neighbors of  $u$  possibly switch in previous updates, the data structures of  $u$  namely  $N_T(u), N_{\bar{T}}(u)$  are not modified. Thus, lists  $N_T(u), N_{\bar{T}}(u)$  are updated and for all high degree neighbors  $w$  of  $u$ ,  $N_X(w), N_{\bar{X}}(w)$  are also updated to reflect  $u$ 's switch. Since  $u$  switched from  $X$  to  $\bar{X}$ , the sizes of lists  $N_X(w), N_{\bar{X}}(w)$  are modified appropriately. For all neighbors  $w \in V_{low}$  of  $u$ , their data structures due to  $u$ 's switch to  $\bar{X}$  are already updated in the call to Algorithm 1. Since  $u$ 's neighbor lists are up-to-date, the counts  $|E(X, T)|, |E(X, \bar{T})|, |E(\bar{X}, T)|, |E(\bar{X}, \bar{T})|$  are correctly updated.

For the case when  $v_i, v_j \in V_{high}$ , Algorithm 1 is called in order to handle the edge update with respect to the induced subgraph  $G_2$ . Let  $u \in V_{high}$  be a vertex which switches and let  $Y, \bar{Y} \in \{T, \bar{T}\}$  be such that  $u \in Y$  before the update and switches to  $\bar{Y}$ . Vertices in  $V_{high}$  are updated to reflect the switch of  $u$  with respect to the cut  $(T, \bar{T})$  during the call to Algorithm 1. Since  $u$  is a high degree vertex, the neighbor lists  $N_S(u), N_{\bar{S}}(u)$  are always up-to-date. Thus, the edge counts  $|E(X, T)|, |E(X, \bar{T})|, |E(\bar{X}, T)|, |E(\bar{X}, \bar{T})|$  are correctly updated. ◀



# Weighted Tiling Systems for Graphs: Evaluation Complexity

C. Aiswarya 

Chennai Mathematical Institute, India  
IRL ReLaX, CNRS, France  
aiswarya@cmi.ac.in

Paul Gastin 

LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France  
paul.gastin@ens-paris-saclay.fr

---

## Abstract

We consider weighted tiling systems to represent functions from graphs to a commutative semiring such as the Natural semiring or the Tropical semiring. The system labels the nodes of a graph by its states, and checks if the neighbourhood of every node belongs to a set of permissible tiles, and assigns a weight accordingly. The weight of a labeling is the semiring-product of the weights assigned to the nodes, and the weight of the graph is the semiring-sum of the weights of labelings. We show that we can model interesting algorithmic questions using this formalism - like computing the clique number of a graph or computing the permanent of a matrix. The evaluation problem is, given a weighted tiling system and a graph, to compute the weight of the graph. We study the complexity of the evaluation problem and give tight upper and lower bounds for several commutative semirings. Further we provide an efficient evaluation algorithm if the input graph is of bounded tree-width.

**2012 ACM Subject Classification** Theory of computation → Quantitative automata

**Keywords and phrases** Weighted graph tiling, tiling automata, Evaluation, Complexity, Tree-width

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.34

**Related Version** An extended version of the paper is available at <https://arxiv.org/abs/2009.14542>

**Funding** Supported by IRL ReLaX.

*C. Aiswarya*: Supported by DST Inspire.

## 1 Introduction

Weighted automata have been classically studied over words, as they naturally extend automata from representing languages to representing functions from words to a semiring.

We are interested in finite state formalisms for representing functions from *graphs* to a semiring. Many natural algorithmic questions on graphs are about computing a function, such as the clique number, weight of the shortest path etc. It is interesting to see if one can design weighted automata to model such problems. Further can one design efficient algorithms for problems modeled by such weighted automata?

We study weighted tiling systems (WTS), a variant of the weighted graph automata of Droste and Dück [10], motivated by the graph acceptors of Thomas [24]. This subsumes many quantitative models that have been studied on words, trees [13, 14], nested words [22], pictures [17], Mazurkiewicz traces [11, 23, 5], etc. The reader is referred to the handbook [12] for more details and references. Many of these works are mainly interested in expressivity questions, and show that the model has good expressive power. The model is also easy to understand as it is formulated in terms of tiling/colouring respecting local constraints. We reiterate the expressivity by modeling computational problems on graphs using this model.



© C. Aiswarya and Paul Gastin;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 34; pp. 34:1–34:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our focus is on the computational complexity of the evaluation problem. It is closer in spirit to [18] which provides an efficient evaluation algorithm for weighted pebble automata on words.

We show that many algorithmic questions, like computing the clique number, computing the permanent of a matrix, or counting-variants of SAT, can be naturally modeled using this formalism. We investigate the computational complexity of the evaluation problem and obtain tight upper- and lower-bounds for various semirings.

To give more details, a WTS has a finite number of states and a run labels the vertices of a graph with states. The tiles (analogous to transitions) observe the neighbourhood of a vertex under the labeling, and assign a weight accordingly. The weight of the run is the semiring-product of the weights thus assigned, and the weight assigned to a graph is the semiring-sum of the weights of the runs. We only consider commutative semirings and hence the order in which the product is taken does not matter.

The evaluation problem is to compute the weight of an input graph in an input WTS. We study the computational complexity of this problem for various semirings. Over Natural semiring and non-negative rationals, the problem is shown to be #P-complete. Over integers and rationals the problem is GapP-complete. Over tropical semirings –  $(\mathbb{N}, \max, +)$ ,  $(\mathbb{Z}, \max, +)$ ,  $(\mathbb{N}, \min, +)$ ,  $(\mathbb{Z}, \min, +)$  – the problem is  $\text{FP}^{\text{NP}[\log]}$  complete.

We further consider the evaluation problem for graphs of bounded tree-width and show that they are computable in time polynomial in the WTS and linear in the graph. Bounded tree-width captures a variety of formal models of concurrent and infinite state systems such as Mazurkiewicz traces, nested words, and decidable under-approximations of message passing automata or multi-pushdown automata [21, 1, 2, 9, 4].

Even though our focus is evaluation, and not expressiveness of the model, we get a deep insight into the modeling power of this formalism through the upper and lower complexity bounds. For instance, we cannot polynomially encode the traveling salesman problem (lower bound  $\text{FP}^{\text{NP}}$ ) in our formalism over tropical semiring (upper bound  $\text{FP}^{\text{NP}[\log]}$ ) unless the polynomial hierarchy collapses [19].

## 2 Model

First we will fix the notations for semirings, graphs and then introduce the WTS formally.

**Preliminaries.** Let  $\mathbb{N}$  denote the set of natural numbers including 0,  $\mathbb{Z}$  the integers, and  $\mathbb{Q}$  the rationals.

Let  $A = \{a_1, \dots, a_n\}$  and  $B$  be two sets. We sometimes write a function  $f: A \rightarrow B$  explicitly by listing the image of each element:  $f = [a_1 \mapsto f(a_1), \dots, a_n \mapsto f(a_n)]$ . The set of all functions from  $A$  to  $B$  is denoted  $B^A$ . If  $A$  is  $\emptyset$  then the only relation (and hence function) from  $A$  to  $B$  is  $\emptyset$ . We denote this trivial empty function by  $f_\emptyset$ .

Let  $M$  be a non-deterministic Turing machine. The number of accepting runs of  $M$  on an input  $x$  is denoted  $\#M(x)$ , and the number of rejecting runs of  $M$  on  $x$  is denoted  $\#\bar{M}(x)$ .

A semiring is an algebraic structure  $\mathbb{S} = (S, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$  where  $S$  is a set,  $\oplus$  and  $\otimes$  are two binary operations on  $S$ ,  $(S, \oplus, 0_{\mathbb{S}})$  is a commutative monoid,  $(S, \otimes, 1_{\mathbb{S}})$  is a monoid,  $\otimes$  distributes over  $\oplus$ ,  $0_{\mathbb{S}}$  is an annihilator for  $\otimes$ . A semiring is *commutative* if  $\otimes$  is commutative.

Examples are **Boolean** =  $(\{0, 1\}, \vee, \wedge, 0, 1)$ , **Natural** =  $(\mathbb{N}, +, \times, 0, 1)$ , **Integer** =  $(\mathbb{Z}, +, \times, 0, 1)$ , **Rational** =  $(\mathbb{Q}, +, \times, 0, 1)$  and **Rational<sup>+</sup>** =  $(\mathbb{Q}_{\geq 0}, +, \times, 0, 1)$ . Further examples are tropical semirings: **max-plus- $\mathbb{N}$**  =  $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ , **max-plus- $\mathbb{Z}$**  =  $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$ , **min-plus- $\mathbb{N}$**  =  $(\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$  and **min-plus- $\mathbb{Z}$**  =  $(\mathbb{Z} \cup \{+\infty\}, \min, +, +\infty, 0)$ . We will consider only these semirings in this paper. Note that all these semirings are commutative.

**Graphs.** We consider graphs with different sorts of edges. For example, a grid will have horizontal successor edges, and vertical successor edges. A binary tree will have left-child relations and right-child relations. Message sequence charts will have process-successor relations and message send-receive relations. These graphs have bounded degree, and for each sort of edge, a vertex will have at most one outgoing/incoming edge of that sort<sup>1</sup>. Our definition of graphs below allows to capture such graph classes.

Let  $\Gamma$  be a finite set of edge names, and let  $\Sigma$  be a finite set of node labels. A  $(\Gamma, \Sigma)$ -graph  $G = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda)$  has a finite set of vertices  $V$ , an edge relation  $E_\gamma \subseteq V \times V$  for every  $\gamma \in \Gamma$ , and a mapping  $\lambda: V \rightarrow \Sigma$  assigning a label from  $\Sigma$  to each vertex  $v \in V$ . The graphs we consider will have at most one outgoing edge and at most one incoming edge for every edge name. That is, for each  $\gamma \in \Gamma$ , for all  $v \in V$ ,  $|\{u \mid (v, u) \in E_\gamma\}| \leq 1$  and  $|\{u \mid (u, v) \in E_\gamma\}| \leq 1$ .

The type of a vertex is determined by the set of names of incoming edges and the set of names of outgoing edges. For example, the root of a tree has no incoming left-child or right-child edges and leaves of a tree have no outgoing left- or right-child. A type  $\tau = (\Gamma_{\text{in}}, \Gamma_{\text{out}})$  indicates that the set of incoming (resp. outgoing) edge names is  $\Gamma_{\text{in}}$  (resp.  $\Gamma_{\text{out}}$ ). Let  $\text{Types} = 2^\Gamma \times 2^\Gamma$  be the set of all types. We define  $\text{type}: V \rightarrow \text{Types}$  and use  $\text{type}(v)$  to denote the type of vertex  $v$ .

► **Remark 1.** Even though we consider only bounded degree graphs, we are able to model graph functions on arbitrary graphs (even edge weighted) as illustrated in the examples below. Basically an arbitrary graph is input via its adjacency matrix, which is naturally a grid, a special case of the graphs that we can handle. We can even model problems on arbitrary graphs with edge weights.

**A weighted Tiling System.** is a finite state mechanism for defining functions from a class of graphs to a weight domain. It has a finite set of states and a set of permissible tiles for each type of vertices. Formally, a *weighted tiling system* (WTS) over  $(\Gamma, \Sigma)$ -graphs and a semiring  $\mathbb{S} = (S, \oplus, \otimes, 0_{\mathbb{S}}, 1_{\mathbb{S}})$  is a tuple  $\mathcal{T} = (Q, \Delta, \text{wgt})$  where

- $Q$  is the finite set of states,
- $\Delta = \bigcup_{\tau \in \text{Types}} \Delta_\tau$  – for a type  $\tau = (\Gamma_{\text{in}}, \Gamma_{\text{out}}) \in \text{Types}$ , the set  $\Delta_\tau \subseteq Q^{\Gamma_{\text{in}}} \times Q \times \Sigma \times Q^{\Gamma_{\text{out}}}$  gives the set of permissible tiles of type  $\tau$ ,
- $\text{wgt}: \Delta \rightarrow S$ , assigns a weight for each tile.

A run  $\rho$  of  $\mathcal{T}$  on a graph  $G = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda)$  is a labeling of the vertices by states that conforms to  $\Delta$ . Given a labeling  $\rho: V \rightarrow Q$ , for a vertex  $v \in V$  with  $\text{type}(v) = (\Gamma_{\text{in}}, \Gamma_{\text{out}})$  we define the tile of  $v$  wrt.  $\rho$  to be  $\text{tile}_\rho(v) = (f_{\text{in}}, \rho(v), \lambda(v), f_{\text{out}})$  where  $f_{\text{in}}: \Gamma_{\text{in}} \rightarrow Q$  is given by  $\gamma \mapsto \rho(u)$  if  $(u, v) \in E_\gamma$  and  $f_{\text{out}}: \Gamma_{\text{out}} \rightarrow Q$  is given by  $\gamma \mapsto \rho(u)$  if  $(v, u) \in E_\gamma$ . A labeling  $\rho: V \rightarrow Q$  is a *run* if for each  $v \in V$ ,  $\text{tile}_\rho(v) \in \Delta_{\text{type}(v)}$ .

The *weight of a run*  $\rho$ , denoted  $\text{wgt}(\rho)$ , is the product of the weights of the tiles in  $\rho$ . With commutative semirings, we do not need to specify an order for this product. The value  $[[\mathcal{T}]](G)$  computed by  $\mathcal{T}$  for a graph  $G$  is the sum of the weights of the runs. That is,

$$[[\mathcal{T}]](G) = \bigoplus_{\rho \mid \rho \text{ is a run of } \mathcal{T} \text{ on } G} \text{wgt}(\rho) \qquad \text{wgt}(\rho) = \bigotimes_{v \in V} \text{wgt}(\text{tile}_\rho(v)).$$

► **Remark 2.** The WTS is a variant of the weighted graph automata (WGA) of [10]. There are two main differences. First, WGA admits tiles of bigger radius and the tile size is a

<sup>1</sup> This choice is mainly for notational convenience, and is not really a restriction, provided we consider only bounded degree graphs. Another option would be to enumerate the neighbours in some order and address a neighbour as the  $i$ th incoming/outgoing neighbour.

### 34:4 Weighted Tiling Systems for Graphs: Evaluation Complexity

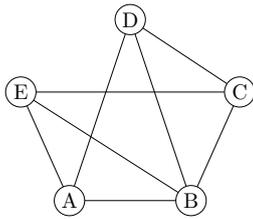
parameter. This is not more powerful, as it can be realized with immediate neighborhood tiles like in WTS. Second, WGA allows occurrence constraints. We discuss this in more detail in Section 5.

We give some examples of WTS below, which will also serve as reductions proving complexity lower-bounds in Section 3.

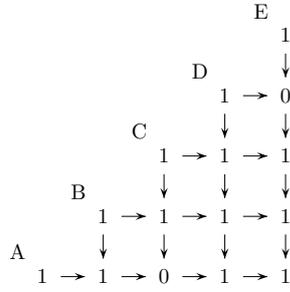
► **Example 3 (A WTS to compute the clique number of a graph).** The *clique number* of a graph is the size of the largest clique in the graph.

The graphs on which we want to compute the clique number have unbounded degrees indeed. In our setting we consider only bounded degree graphs. Hence we need to encode any arbitrary graph as a bounded degree graph. One way to do that is to consider the adjacency matrix and represent this matrix using a grid graph.

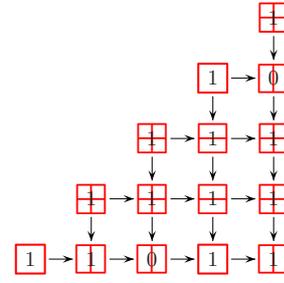
For the particular case of clique number, our input is an undirected graph, so we will consider a lower-right triangular matrix in a lower-right triangular grid graph. For this we let  $\Gamma = \{\rightarrow, \downarrow\}$  and  $\Sigma = \{0, 1\}$ . The labels of all diagonal vertices are 1. A graph is depicted in Figure 1 and its lower-right triangular adjacency matrix is depicted in Figure 2.



■ **Figure 1** A graph



■ **Figure 2** The lower-right triangular adjacency matrix of the graph of Figure 1 as a grid graph



■ **Figure 3** A run. Three tiles B, C and E gets weights 1, and hence the weight of this run is 3.

We will now construct a WTS over the tropical semiring  $\max\text{-plus-}\mathbb{N}$  that computes the clique number on a lower triangular grid graph. The run of the WTS will guess a subset of vertices of the original graph (corresponds to labeling some diagonal elements with state  $\boxplus$ ) and checks that there is an edge between every pair of these (corresponds to checking the label is 1, if the row and column start in a  $\boxplus$ -labeled vertex). The weight of such a run will be the size of the subset, and the max over all the runs gives us the clique number as required.

Let  $Q = \{\boxplus, \boxminus, \boxtimes, \boxdot\}$ . A run will label a subset of diagonal vertices with  $\boxplus$ . A vertex is labeled with  $\boxminus$  (resp.  $\boxtimes, \boxdot$ ) if its column (resp. row, both) starts in a vertex labeled  $\boxplus$ . In addition a vertex may get state  $\boxplus$  only if its label is 1. All other vertices get state  $\boxdot$ . A run on the graph in Figure 2 is depicted in Figure 3.

Tiles for diagonal vertices are given by  $\Delta_{(\emptyset, \Gamma_{\text{out}})} = \{(f_{\emptyset}, \boxdot, 1, f_{\text{out}}), (f_{\emptyset}, \boxplus, 1, f_{\text{out}})\}$ . For an inside vertex we have ( $f_{\text{out}}$  being arbitrary in all tuples):

$$\begin{aligned} \Delta_{(\{\rightarrow, \downarrow\}, \Gamma_{\text{out}})} = & \{(f_{\text{in}}, \boxdot, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\rightarrow) \in \{\boxminus, \boxdot\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxdot\}\} \\ & \cup \{(f_{\text{in}}, \boxplus, 1, f_{\text{out}}) \mid f_{\text{in}}(\rightarrow) \in \{\boxplus, \boxtimes\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxminus\}\} \\ & \cup \{(f_{\text{in}}, \boxtimes, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\rightarrow) \in \{\boxplus, \boxtimes\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxdot\}\} \\ & \cup \{(f_{\text{in}}, \boxdot, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\rightarrow) \in \{\boxminus, \boxdot\}, f_{\text{in}}(\downarrow) \in \{\boxplus, \boxtimes\}\}. \end{aligned}$$

The weight of a tile of the form  $(f_{\emptyset}, \boxplus, 1, f_{\text{out}})$  is 1. Notice that only the diagonal vertices labeled  $\boxplus$  will get such a tile. The weight of all other tiles is 0. Thus the weight of a run is the number of diagonal vertices labeled  $\boxplus$  - which corresponds to a subset of vertices inducing a clique. The maximum weight across different runs will compute the clique number as required. ◀

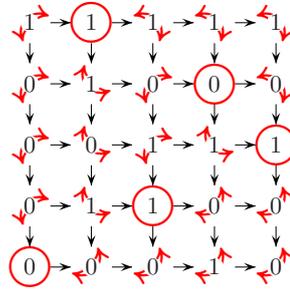
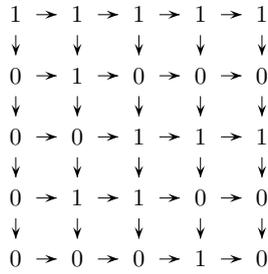
► **Example 4** (A WTS to compute the permanent of a (0,1)-matrix). We will model (0,1)-matrices as (0,1)-labelled grids. As in Example 3, we let  $\Gamma = \{\rightarrow, \downarrow\}$  and  $\Sigma = \{0, 1\}$ . A  $5 \times 5$  (0,1)-matrix as a grid graph is illustrated in Figure 4.

We will define a WTS  $\mathcal{T}$  on such graphs over **Natural** such that  $[[\mathcal{T}]](G)$  is the permanent of the 0,1 matrix  $A$  represented by  $G$ . In each run exactly one vertex in each row and each column will be circled – representing one permutation  $\sigma$  of  $\{1, \dots, n\}$  if  $G$  is an  $n \times n$  grid. The weight of the tile on the circled vertex will be the vertex label (0 or 1) interpreted as an integer. Every other tile will have weight 1. Thus the weight of a run will be  $\prod_i A(i, \sigma(i))$  where  $\sigma$  is the permutation represented by the run. Finally the value of a graph  $G$  representing an  $n \times n$  (0,1)-matrix  $A$  will be  $\sum_{\sigma} \prod_i A(i, \sigma(i))$  which is its permanent.

The WTS  $\mathcal{T}$  has five states:  $Q = \{\circlearrowleft, \uparrow, \swarrow, \searrow, \circlearrowright\}$ . We will define tiles so as to accept only the labeling reflecting the following:

- a vertex labeled  $\circlearrowleft$  means it is the circled vertex in its row and column,
- a vertex  $v$  labeled  $\uparrow$  means that the circled vertex in its column is upward of  $v$ , and the circled vertex in its row is to the right of  $v$ ,
- similarly for other states  $\swarrow, \searrow, \circlearrowright$ .

The tiles are given formally below. The weight function **wgt** assigns weight 0 to any tile labeling a 0-labeled node with  $\circlearrowleft$ . The weight of all other tiles is 1. A run of this WTS is illustrated in Figure 5.



■ **Figure 4** A  $5 \times 5$  (0,1)-matrix as a grid graph    ■ **Figure 5** A run of the WTS  $\mathcal{T}$  on the graph in Fig. 4. It has weight 0 as two tiles have **wgt** 0.

We now describe the tiles formally. For the top-left vertex we have

$$\Delta_{(\emptyset, \{\rightarrow, \downarrow\})} = \{(f_{\emptyset}, \circlearrowleft, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{out}}(\rightarrow) = \searrow, f_{\text{out}}(\downarrow) = \uparrow\} \cup \{(f_{\emptyset}, \swarrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{out}}(\rightarrow) \in \{\swarrow, \circlearrowleft\}, f_{\text{out}}(\downarrow) \in \{\swarrow, \circlearrowleft\}\}$$

The tiles for other corner vertices are analogous. For the left border vertices we have  $\Delta_{(\{\downarrow\}, \{\rightarrow, \downarrow\})} =$

$$\{(f_{\text{in}}, \circlearrowleft, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) = \swarrow, f_{\text{out}}(\rightarrow) \in \{\searrow, \swarrow\}, f_{\text{out}}(\downarrow) = \uparrow\} \cup \{(f_{\text{in}}, \swarrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) = \swarrow, f_{\text{out}}(\rightarrow) \in \{\searrow, \swarrow, \circlearrowleft\}, f_{\text{out}}(\downarrow) \in \{\swarrow, \circlearrowleft\}\} \cup \{(f_{\text{in}}, \uparrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\searrow, \circlearrowleft\}, f_{\text{out}}(\rightarrow) \in \{\searrow, \swarrow, \circlearrowleft\}, f_{\text{out}}(\downarrow) = \uparrow\}$$

## 34:6 Weighted Tiling Systems for Graphs: Evaluation Complexity

The tiles for other border vertices are analogous. For an interior vertex, we have

$$\begin{aligned} \Delta_{(\{\rightarrow, \downarrow\}, \{\rightarrow, \downarrow\})} = & \{(f_{\text{in}}, \circ, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\leftarrow, \curvearrowright\}, f_{\text{in}}(\rightarrow) \in \{\leftarrow, \curvearrowright\}, \\ & f_{\text{out}}(\rightarrow) \in \{\leftarrow, \leftarrow\}, f_{\text{out}}(\downarrow) \in \{\leftarrow, \leftarrow\}\} \\ \cup & \{(f_{\text{in}}, \curvearrowright, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\leftarrow, \curvearrowright\}, f_{\text{in}}(\rightarrow) \in \{\leftarrow, \curvearrowright\}, \\ & f_{\text{out}}(\rightarrow) \in \{\leftarrow, \circ, \curvearrowright\}, f_{\text{out}}(\downarrow) \in \{\leftarrow, \circ, \curvearrowright\}\} \\ \cup & \{(f_{\text{in}}, \leftarrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\leftarrow, \circ, \leftarrow\}, f_{\text{in}}(\rightarrow) \in \{\leftarrow, \curvearrowright\}, \\ & f_{\text{out}}(\rightarrow) \in \{\leftarrow, \circ, \curvearrowright\}, f_{\text{out}}(\downarrow) \in \{\leftarrow, \leftarrow\}\} \\ \cup & \{(f_{\text{in}}, \leftarrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\leftarrow, \circ, \leftarrow\}, f_{\text{in}}(\rightarrow) \in \{\leftarrow, \circ, \leftarrow\}, \\ & f_{\text{out}}(\rightarrow) \in \{\leftarrow, \leftarrow\}, f_{\text{out}}(\downarrow) \in \{\leftarrow, \leftarrow\}\} \\ \cup & \{(f_{\text{in}}, \leftarrow, b, f_{\text{out}}) \mid b \in \{0, 1\}, f_{\text{in}}(\downarrow) \in \{\leftarrow, \curvearrowright\}, f_{\text{in}}(\rightarrow) \in \{\leftarrow, \circ, \leftarrow\}, \\ & f_{\text{out}}(\rightarrow) \in \{\leftarrow, \leftarrow\}, f_{\text{out}}(\downarrow) \in \{\leftarrow, \circ, \curvearrowright\}\} \end{aligned}$$

Finally, we describe the weight function  $\text{wgt}$ . The weight of a tile of the form  $(f_{\text{in}}, \circ, 0, f_{\text{out}})$  is 0. The weight of all other tiles is 1.  $\blacktriangleleft$

► **Example 5** (Permanent of matrix with entries from  $\mathbb{N}$ ). The purpose of this example is to illustrate that it is possible to encode natural numbers, which may appear as matrix entries or edge weights, also as bounded degree graphs with a fixed alphabet  $\Sigma$ .

A length  $k$  bit string  $b_{k-1} \cdots b_1 b_0$  where  $b_i \in \{0, 1\}$  for all  $0 \leq i < k$ , is represented by a path graph of length  $k$ . The vertices of this path graph are labelled with 1 or 0 to indicate the value of the bit, and the edges are labeled  $<$ . We describe a WTS on such path graphs whose computed weight is the binary number  $\sum_i b_i 2^i$ . The WTS guesses a prefix ending with label 1. All the nodes in the prefix take state  $q_0$  and all nodes after the prefix may take the two states  $q_1$  or  $q_2$ . The weight of all tiles is 1. The number of runs is  $\sum_{i:b_i=1} 1^{k-i} \times 2^i = \sum_i b_i 2^i$ .

As before, we will have an  $n \times n$  grid graph to represent the matrix, but the vertices of the grid graph take a neutral label, say  $X$ . A path graph originates from every vertex of the grid graph indicating the entry of the matrix at that cell. Now, to compute the permanent, the path graphs starting from a circled vertex can start the WTS described in the previous paragraph. All other path graphs vertices can be labeled only by a special state  $q_4$ . The weights of all permissible tiles are 1. The weight computed by one permutation will indeed be the product of the entries. This crucially depends on the distributivity of the semiring. Thus, this WTS computes the permanent of an arbitrary matrix with entries in  $\mathbb{N}$ .  $\blacktriangleleft$

**Evaluation problem (Eval).** is to compute  $[[\mathcal{T}]](G)$ , given the following input:

- $\mathcal{T}$  : a WTS over  $(\Gamma, \Sigma)$ -graphs and a semiring  $\mathbb{S}$ , and
- $G$  : a  $(\Gamma, \Sigma)$ -graph.

We study the complexity of this problem in Section 3, for various semirings. We provide an efficient algorithm for this problem in the case of bounded tree-width graphs in Section 4.

### 3 Evaluation complexity: Arbitrary graphs

Recall that we only consider the boolean semiring, the counting semirings over  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$  or  $\mathbb{Q}_{\geq 0}$  and the tropical semirings over  $\mathbb{N}$  or  $\mathbb{Z}$ .

Given a WTS  $\mathcal{T}$  and a graph  $G$ , we can compute  $[[\mathcal{T}]](G)$  in polynomial space as follows. Initialise the current aggregate to  $0_{\mathbb{S}}$ . Enumerate in lexicographic order through the different

labelings of the vertices of  $G$  with states of  $\mathcal{T}$ . For each labeling, if it conforms to  $\Delta$ , compute its weight and add to the current aggregate. Thus **Eval** belongs to **FPSpace** – the set of functions computable in polynomial space.

► **Theorem 6.** *Problem Eval is in FPSpace.*

However, for particular semirings the complexity is different as stated in the following subsections.

### 3.1 $(+, \times)$ -semirings

► **Theorem 7.** *The evaluation problem is #P-complete over Natural, and non-negative Rational. It is GapP-complete over Integer and Rational.*

The upper bounds hold for arbitrary graphs, and the lower bounds hold for the special case of grids. The weights can be assumed to be given in binary.

A function  $f$  is in #P if there is an NP machine  $M$  such that  $f(x) = \#M(x)$ . That is, it denotes the set of function problems that correspond to counting the number of accepting paths in a non-deterministic polynomial time turing machine. Computing the permanent of a  $(0,1)$ -matrix is a #P-complete problem [25], and hence the #P-hardness claimed above follows from Example 4. We give an alternate hardness proof by a reduction from #-CNF-SAT.

A function  $f(x)$  is in GapP if there is a non-deterministic polynomial time turing machine  $M$  such that  $f(x) = \#M(x) - \#\overline{M}(x)$ . GapP is also the closure of #P under subtraction.

Most of this subsection is devoted to the proof of Theorem 7. First we give the non-deterministic Turing machines realising the upper bounds for **Natural** and **Integer**. After that we give reductions from respective counting versions of SAT to prove the lower bounds. The case of **Rational** is finally considered.

**The Turing Machine  $\mathcal{M}$  such that  $\#\mathcal{M}(\mathcal{T}, G) = \llbracket \mathcal{T} \rrbracket(G)$ .** We describe a non-deterministic polynomial time turing machine  $\mathcal{M}$  that takes as input a WTS  $\mathcal{T}$  over **Natural** with weights given in binary, and a graph  $G$ . The number of accepting runs  $\#\mathcal{M}(\mathcal{T}, G) = \llbracket \mathcal{T} \rrbracket(G)$ . We assume the states, weights etc. are given by some standard encoding.

The turing machine  $\mathcal{M}$  non-deterministically guesses a labeling of the vertices of  $G$  by the states of  $\mathcal{T}$ . Then it computes the product  $w$  of the weights of the tiles in the guessed tiling and writes it in binary (MSB on the left) in a different tape. Computing the product can be done in time polynomial in  $|G|$  and  $\log(k)$  where  $k = \max\{x \mid x \text{ is a weight of some tile of } \mathcal{T}\}$ .

Afterwards it enters a phase which will have exactly  $w$  different accepting branches. Simply decrementing the value while it is positive, and non-deterministically accepting at any step will have  $w$  accepting branches, but the running time is exponential. We want the machine to run in polynomial time. Hence we implement this phase similar to Example 5. It runs in  $\mathcal{O}(|w|)$  steps as we detail below.

$\mathcal{M}$  scans  $w$  from left to right starting in some state  $q$ . While in state  $q$  and the current cell is labeled 0 it moves right. If in state  $q$  and the current cell is labelled 1 it moves right and non-deterministically stays in state  $q$  or enters one of the two special states  $q_0$  or  $q_1$ . When it is in state  $q_0$  or  $q_1$  and the current cell is labelled with 0 or 1, it will move right and non-deterministically chose either  $q_0$  or  $q_1$ . Finally, When in state  $q_0$  or  $q_1$  and the current cell is *blank* (i.e., the scan of  $w$  is over), then  $\mathcal{M}$  accepts. Thus if the  $i$ th bit from the right of  $w$  is labeled 1, then  $\mathcal{M}$  can have  $2^i$  accepting runs if it moved from state  $q$  to  $q_0$  or  $q_1$  when

reading this bit. Switching from state  $q$  can occur at any 1-labelled cell, and hence  $\mathcal{M}$  will have  $w$  many accepting runs.

The machine  $\mathcal{M}$  non deterministically picks a labeling at first, and hence the total number of accepting runs  $\#\mathcal{M}(\mathcal{T}, G) = \llbracket \mathcal{T} \rrbracket(G)$ . With this we prove the #P upper bound for Natural.

**The Turing Machine  $\mathcal{M}'$  such that  $\#\mathcal{M}'(\mathcal{T}, G) - \#\overline{\mathcal{M}'}(\mathcal{T}, G) = \llbracket \mathcal{T} \rrbracket(G)$**  This is similar to the machine  $\mathcal{M}$  above. There are two differences. The machine  $\mathcal{M}'$  still guesses a labeling of vertices of  $G$  with states of  $\mathcal{T}$  over Integer and computes the weight  $w$ . If  $w$  is positive, it proceeds exactly as  $\mathcal{M}$  does to produce  $w$  accepting runs. If the weight  $w$  is negative, the machine  $\mathcal{M}'$  proceeds analogously but with states  $q'$ ,  $q'_0$  and  $q'_1$  instead. If the machine is in state  $q'_0$  or  $q'_1$  with current cell *blank* then it rejects instead of accepting. The second difference is for blocked runs (e.g., if the guessed labeling of vertices of  $G$  by states of  $\mathcal{T}$  is not a valid tiling, or if at the end the machine is still in state  $q$  or  $q'$  with current cell *blank*). In such a case,  $\mathcal{M}'$  will non-deterministically proceed to either accept or reject. Thus the net difference between accepting runs and rejecting runs is kept intact and  $\#\mathcal{M}'(\mathcal{T}, G) - \#\overline{\mathcal{M}'}(\mathcal{T}, G) = \llbracket \mathcal{T} \rrbracket(G)$ . This proves the GapP upper bound for Integer.

**Encoding a CNF formula  $\varphi$  in a grid  $G_\varphi$ .** Given a CNF formula  $\varphi$  with  $n$  variables and  $m$  clauses, we encode it in an  $n \times m$  grid with node labels  $\{p, n, \star\}$ . If the node  $(i, j)$  is labeled by  $p$  (resp.  $n$ ) it means that the  $i$ th variable appears in  $j$ th clause positively (resp. negatively). The node  $(i, j)$  is labeled  $\star$  if the  $i$ th variable does not occur in the  $j$ th clause.

**A WTS  $\mathcal{T}^\#$  over Natural for counting  $\#\varphi$ .** Recall that  $\#\varphi$  is the number of satisfying assignments for the formula  $\varphi$ . We assume input to the WTS  $\mathcal{T}^\#$  is given as  $G_\varphi$  – a  $\{p, n, \star\}$ -labeled grid encoding a CNF formula.

A state of  $\mathcal{T}^\#$  is a pair from  $\{q_{\text{true}}, q_{\text{false}}\} \times \{q'_{\text{true}}, q'_{\text{false}}\}$ . The first part of a state indicates a truth assignment with  $q_{\text{true}}$  and  $q_{\text{false}}$ . The allowed tiles make sure that in this part the truth assignment remains the same along a row. The second part of a state indicates with  $q'_{\text{true}}$  and  $q'_{\text{false}}$  the partial evaluation of the formula. A  $p$ -labeled node which is assigned  $q_{\text{true}}$  from the first part, and an  $n$ -labeled node which is assigned  $q_{\text{false}}$  from the first part gets the value  $q'_{\text{true}}$  in the second part of the state (call this condition A for future reference). Further all the successor nodes in the column of the  $q'_{\text{true}}$  labeled node also gets the value  $q'_{\text{true}}$ , except for the nodes in the last row. For the nodes in the last row, it gets the value  $q'_{\text{true}}$  if the left neighbour is labeled  $q'_{\text{true}}$  (assume this is satisfied if the left neighbour does not exist), and a) if it satisfies condition A or b) if the node above is labeled  $q'_{\text{true}}$ . Otherwise the nodes get the value  $q'_{\text{false}}$ . The second part of a state labeling a node  $(n, j)$  in the last row indicates the evaluation of the prefix of the formula until the  $j$ th clause.

The tiles capture the description above. The weight of all tiles is 1, except for the tile labeling the last node  $(n, m)$ . If it is labeled  $(-, q'_{\text{true}})$  then the weight is 1, otherwise it is 0. The value  $\llbracket \mathcal{T}^\# \rrbracket(G_\varphi) = \#\varphi$ , the number of satisfying assignments.

This proves the #P lower bound for Natural. As alluded to earlier, the permanent computation (Example 4) gives an alternate lower bound proof.

**A WTS  $\mathcal{T}^{\text{gap}}$  over Integer for counting  $\#\varphi_1 - \#\varphi_2$ .** We will reduce the GapP-complete problem of computing  $\#\varphi_1 - \#\varphi_2$ , where  $\varphi_1$  and  $\varphi_2$  are input CNF formulas on the same set of  $n$  variables with  $m_1$  and  $m_2$  clauses respectively. We represent the input in an  $n \times (m_1 + m_2)$  grid by putting  $G_{\varphi_1}$  and  $G_{\varphi_2}$  side by side. The node labels contain a special tag  $i \in \{1, 2\}$  to

indicate that it comes from  $G_{\varphi_i}$ . The WTS  $\mathcal{T}^{\text{GapP}}$  will ensure that rows are of the form  $1^*2^*$  and columns are of the form  $1^*$  or  $2^*$ . In a run it evaluates either  $\varphi_1$  or  $\varphi_2$  similar to  $\mathcal{T}^\#$ . If it is evaluating  $\varphi_i$  all nodes with the tag  $3-i$  gets a special state  $q_{\text{skip}}$ . The weight of all tiles is 1, except for the tile labeling the nodes  $(n, m_1)$  and  $(n, m_1 + m_2)$ . If the node  $(n, m_1)$  is labeled  $(-, q'_{\text{true}})$  or  $q_{\text{skip}}$  then the weight is 1, otherwise it is 0. If the node  $(n, m_1 + m_2)$  is labeled  $(-, q'_{\text{true}})$  (resp.  $q_{\text{skip}}$ ) then the weight is  $-1$  (resp. 1), otherwise it is 0.

**Rational.** We will use counting reduction from Rational (resp. non-negative Rational) to the evaluation problem over Integer (resp. Natural) in order to prove the upper bounds. First we will transform an input  $(\mathcal{T}, G)$  of the evaluation problem over Rational (resp. non-negative Rational) to an input  $(\mathcal{T}', G)$  over Integer (resp. Natural). In  $\mathcal{T}'$  we will multiply the weight of a tile by  $\ell$  - the lcm of the denominators appearing in the weights of any tile of  $\mathcal{T}$ . The multiplication can be performed in time polynomial. Now  $\mathcal{T}'$  is a WTS over Integer (resp. Natural), and following the GapP procedure (resp. #P procedure) we compute  $[[\mathcal{T}']](G)$ . Now, we transform the output back to the required output over Rational (resp. non-negative Rational) by dividing with  $\ell^{|V_G|}$ . That is,  $\text{Eval}(G, \mathcal{A}) = \frac{\text{Eval}(G, \mathcal{A}')}{\ell^{|V_G|}}$ .

Notice that we allow the weights to be given in binary. The lcm  $\ell$  and  $\ell^{|V_G|}$  can be computed in polynomial time. The counting reduction is hence polynomial. This proves the upper bounds.

The GapP-hardness (resp. #P-hardness) follows because Integer (resp. Natural) is a special case of Rational (resp. non-negative Rational).

### 3.2 Boolean semiring

Note that the evaluation problem Eval over Boolean is in fact the classical Membership problem (denoted Membership) and is indeed a decision problem. We can check in NP whether the value is 1 (witnessed by the NP machine  $\mathcal{M}$ , if the input is assumed to be over Boolean then  $\times$  serve as  $\wedge$ ). It is also NP-hard by a simple reduction from CNF SAT (witnessed by  $\mathcal{T}^\#$  interpreted over Boolean).

► **Theorem 8.** *Membership is NP-complete.*

### 3.3 Tropical semirings

► **Theorem 9.** *We assume the weights are given in unary. The evaluation problem over any tropical semiring is  $\text{FP}^{\text{NP}[\log]}$ -complete.*

$\text{FP}^{\text{NP}[\log]}$  is the class of functions computable by a polynomial time turing machine with logarithmically many queries to NP.

**Proof.** We will prove the upper bound for max-plus- $\mathbb{Z}$ . The case of max-plus- $\mathbb{N}$  is subsumed. The cases of min-plus- $\mathbb{N}$  and min-plus- $\mathbb{Z}$  are analogous.

Let  $k$  be the maximal constant and  $\ell$  be the minimal constant (other than  $+/-\infty$ ) appearing in the WTS  $\mathcal{A}$ . The maximum possible weight of a run is  $n \times k$  and the minimum is  $n \times \ell$  where  $n$  is the number of vertices in the input graph. We will do a binary search in the set  $W = \{n \times \ell, \dots, -1, 0, 1, \dots, n \times k\}$  checking if  $[[\mathcal{A}]](G) \geq s$  to find the value of  $[[\mathcal{A}]](G)$ . In each iteration of the binary search, we make an oracle call to the NP machine for  $[[\mathcal{A}]](G) \geq s$ . The number of NP oracle queries is  $\mathcal{O}(\log(n \times k))$  which is only logarithmic in the input size. Recall that the weights are encoded in unary.

Finding the clique number is an  $\text{FP}^{\text{NP}[\log]}$ -complete problem [19]. From Example 3, the lower bound follows. ◀

## 4 Efficient evaluation for bounded tree-width graphs

In this section, we show that the problem `Eval` can be solved efficiently when restricted to graphs of bounded tree-width (the bound is not part of the input). By efficient, we mean time polynomial wrt. the WTS  $\mathcal{T}$  and linear wrt. the graph  $G$  (see Theorems 11 and 13 below). Bounded tree-width covers many graphs used to model behaviours of concurrent or infinite-state systems. For example, it is well-known that words and trees have tree-width 1, nested words used for pushdown systems have tree-width 2, Mazurkiewicz traces describing behaviours of concurrent asynchronous systems with rendez-vous, and most decidable under-approximations of Turing complete models such as multi-pushdown automata, message passing automata with unbounded FIFO channels, etc. [21, 9, 4]. We start by explaining our results for bounded path-width since this is technically simpler. Then we explain how this is extended to bounded tree-width.

### 4.1 Bounded path-width evaluation

**A path decomposition.** of a  $(\Gamma, \Sigma)$ -graph  $G = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda)$ , is a sequence  $V_1, \dots, V_n$  of nonempty subsets of vertices satisfying:

1. for all  $v \in V$ , we have  $v \in V_i$  for some  $1 \leq i \leq n$ ,
2. for all  $(u, v) \in \bigcup_{\gamma \in \Gamma} E_\gamma$ , we have  $u, v \in V_i$  for some  $1 \leq i \leq n$ ,
3. for all  $1 \leq i \leq j \leq k \leq n$ , we have  $V_i \cap V_k \subseteq V_j$ .

The width of the path decomposition is  $\max\{|V_i| - 1 \mid 1 \leq i \leq n\}$ . The path-width of a graph  $G$  is the least  $k$  such that  $G$  admits a path decomposition of width  $k$ .

Words have path-width 1, but trees, nested words, grids have unbounded path-width.

We present below an equivalent definition of path-width which will be convenient to solve the evaluation problem on graphs with bounded path-width. Let  $[k] = \{0, 1, \dots, k\}$ . Graphs over  $(\Gamma, \Sigma)$  of path-width at most  $k$  can be described with words over the alphabet

$$\Omega_k = \{(i, a) \mid i \in [k], a \in \Sigma\} \cup \{\text{Forget}_i \mid i \in [k]\} \cup \{\text{Add}_{i,j}^\gamma \mid i, j \in [k], \gamma \in \Gamma\}$$

The semantics of a word  $\tau \in \Omega_k^*$  is a colored graph  $\{\tau\} = (G_\tau, \chi_\tau)$  where  $G_\tau$  is a  $(\Gamma, \Sigma)$ -labeled graph and  $\chi_\tau: [k] \rightarrow V$  is a partial injective function coloring some vertices of  $G_\tau$ . We say that a color  $i \in [k]$  is *active* in  $\tau$  if it is in the domain of  $\chi_\tau$ . The semantics is defined by induction on the length of  $\tau$ . The semantics of the empty word  $\tau = \varepsilon$  is the empty graph. Assuming that  $\{\tau\} = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda, \chi)$ , we define the effect of appending a new letter to  $\tau$ :  $(i, a)$  adds a new  $a$ -labeled vertex with color  $i$ , provided  $i$  is not active in  $\tau$ ,  $\text{Forget}_i$  removes color  $i$  from the domain of the color map, and  $\text{Add}_{i,j}^\alpha$  adds an  $\alpha$ -labeled edge between the vertices colored  $i$  and  $j$  (if such vertices exist, i.e., if  $i, j$  are active in  $\tau$ ).

We say that a word  $\tau$  over  $\Omega_k$  is *well-formed* if the following conditions are satisfied:

1. if  $\tau' \cdot (i, a)$  is a prefix of  $\tau$  then  $i$  is not active in  $\tau'$ ,
2. if  $\tau' \cdot \text{Forget}_i$  is a prefix of  $\tau$  then  $i$  is active in  $\tau'$ ,
3. if  $\tau' \cdot \text{Add}_{i,j}^\gamma$  is a prefix of  $\tau$  then  $i, j$  are active in  $\tau'$  and the edge labeled  $\gamma$  was not already added in  $\tau'$  between  $\chi_{\tau'}(i)$  and  $\chi_{\tau'}(j)$ .

In the following, a well-formed word over  $\Omega_k$  is called a  $k$ -word. The set  $W_k \subseteq \Omega_k^*$  of  $k$ -words is clearly regular.

- **Lemma 10.** 1. *Given a path decomposition  $V_1, \dots, V_N$  of width at most  $k$  of a  $(\Gamma, \Sigma)$ -graph  $G$ , we can construct in linear time wrt.  $|G|$  a  $k$ -word  $\tau$  such that  $\{\tau\} = (G, \emptyset)$ .*
2. *Given a  $k$ -word  $\tau$ , we can construct a path decomposition of width at most  $k$  of the graph  $G_\tau$  defined by  $\tau$ :  $\{\tau\} = (G_\tau, \chi_\tau)$ .*

**Proof. 1.** We construct by induction a sequence of  $k$ -words  $\tau_\ell$  for  $0 \leq \ell \leq N$  such that  $\{\tau_\ell\} = (G_\ell, \chi_\ell)$  where  $G_\ell$  is the subgraph of  $G = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda)$  induced by the vertices  $V_1 \cup \dots \cup V_\ell$ , and  $\chi_\ell([k]) = V_\ell \cap V_{\ell+1}$  (with  $V_0 = V_{N+1} = \emptyset$ ). We let  $\tau_0 = \epsilon$ .

Let now  $0 \leq \ell < N$  and assume that  $\tau_\ell$  has been constructed. Let  $C_\ell = \text{dom}(\chi_\ell) \subseteq [k]$  be the active colors in  $\tau_\ell$ . By induction, we know that  $|C_\ell| = |V_{\ell+1} \cap V_\ell|$ . Let  $V_{\ell+1} \setminus V_\ell = \{u_1, \dots, u_m\}$ . Since the decomposition is of width at most  $k$ , we have  $|V_{\ell+1}| \leq 1 + k$  and we find  $i_1 < \dots < i_m$  available colors in  $[k] \setminus C_\ell$ . We define  $\tau'_{\ell+1} = \tau_\ell \cdot (i_1, \lambda(u_1)) \cdots (i_m, \lambda(u_m))$ . Let  $D = \{i_1, \dots, i_m\}$  and let  $\{\tau'_{\ell+1}\} = (G', \chi')$ . We have  $\text{dom}(\chi') = C_\ell \cup D$ ,  $\chi'(C_\ell) = V_{\ell+1} \cap V_\ell$  and  $\chi'(D) = V_{\ell+1} \setminus V_\ell$ . For each  $\gamma \in \Gamma$ ,  $i \in C_\ell \cup D$  and  $j \in D$  such that  $(\chi'(i), \chi'(j)) \in E_\gamma$  (resp.  $(\chi'(j), \chi'(i)) \in E_\gamma$ ), we append  $\text{Add}_{i,j}^\gamma$  (resp.  $\text{Add}_{j,i}^\gamma$ ) to the word  $\tau'_{\ell+1}$ . We obtain a  $k$ -word  $\tau''_{\ell+1}$  which defines the subgraph  $G_{\ell+1}$  of  $G$  induced by  $V_1 \cup \dots \cup V_{\ell+1}$ . Notice that, from the third condition of a path decomposition, we have  $V_{\ell+1} \setminus V_\ell = V_{\ell+1} \setminus (V_1 \cup \dots \cup V_\ell)$  and the edges in  $G_{\ell+1}$  which were not already in  $G_\ell$  are between some vertex in  $V_{\ell+1} \setminus V_\ell$  and some vertex in  $V_{\ell+1}$ . Finally, for each  $i \in C_\ell \cup D$  such that  $\chi'(i) \notin V_{\ell+2}$ , we append  $\text{Forget}_i$  to the word  $\tau''_{\ell+1}$ . We obtain the  $k$ -word  $\tau_{\ell+1}$  satisfying our invariant.

Finally, from the invariant we deduce that  $\{\tau_N\} = (G, \emptyset)$ , which concludes the first part of the proof.

**2.** Let  $\tau$  be a  $k$ -word and  $n = |\tau|$  be its length. For  $0 \leq \ell \leq n$ , let  $\tau_\ell$  be the prefix of  $\tau$  of length  $\ell$ . Let  $\{\tau_\ell\} = (G_\ell, \chi_\ell)$  and  $V_\ell = \chi_\ell([k])$  be the subset of vertices which are colored in  $\{\tau_\ell\}$ . We show that  $V_1, \dots, V_n$  is a path decomposition of  $G = G_n = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda)$ .

Let  $u \in V$  be a vertex of  $G$ . For some  $1 \leq \ell \leq n$ , we have  $\tau_\ell = \tau_{\ell-1} \cdot (i, a)$  with  $\chi_\ell(i) = u \in V_\ell$ . This proves that the first condition of a path decomposition is satisfied.

Let  $(u, v) \in E_\gamma$  for some  $\gamma \in \Gamma$ . For some  $1 < \ell < n$ , we have  $\tau_{\ell+1} = \tau_\ell \cdot \text{Add}_{i,j}^\gamma$  with  $\chi_\ell(i) = u$  and  $\chi_\ell(j) = v$ . We deduce that  $u, v \in V_\ell$ , which proves that the second condition of a path decomposition is satisfied.

For the third condition, let  $1 \leq i \leq j \leq m \leq n$  and  $u \in V_i \cap V_m$ . We deduce that for some  $\ell \in [k]$ , we have  $u = \chi_i(\ell) = \chi_m(\ell)$  and that color  $\ell$  was not forgotten between  $\tau_i$  and  $\tau_m$ . Therefore,  $u = \chi_j(\ell) \in V_j$  as desired.  $\blacktriangleleft$

The problem  $k$ -PW-FVal is to compute  $\llbracket \mathcal{T} \rrbracket(G)$ , given a WTS  $\mathcal{T}$  and a  $(\Gamma, \Sigma)$ -graph  $G$  of path-width at most  $k$ .

► **Theorem 11.** *The problem  $k$ -PW-FVal can be solved in linear time wrt. the input graph  $G$  and polynomial time wrt. the input WTS  $\mathcal{T}$ .*

**Proof.** The evaluation algorithm for bounded path-width graphs proceeds in three steps:

1. From the input graph  $G$ , which is assumed to be of path-width at most  $k$ , we compute in linear time a path decomposition  $V_1, \dots, V_n$  using Bodlaender's algorithm [3]. Then, using Lemma 10, we compute in linear time a  $k$ -word  $\tau$  such that  $\{\tau\} = (G, \emptyset)$ .
2. By Lemma 12 below, we construct in time polynomial in  $\mathcal{T}$  a weighted word automaton  $\mathcal{B}_k$  which is equivalent to  $\mathcal{T}$  on graphs of path-width at most  $k$ .
3. We compute  $\llbracket \mathcal{B}_k \rrbracket(\tau)$ . It is well-known that the value of a weighted word automaton  $\mathcal{B}$  on a given word  $w$  can be computed in time  $\mathcal{O}(|\mathcal{B}| \cdot |w|)$  assuming that sum and product in the semiring take constant time.  $\blacktriangleleft$

A weighted word automaton over alphabet  $\Sigma$  is usually given as a tuple  $\mathcal{B} = (Q, T, I, F, \text{wgt})$  where  $I, F \subseteq Q$  are the subsets of initial and final states,  $T \subseteq Q \times \Sigma \times Q$  defines the transitions and  $\text{wgt}: T \rightarrow S$  gives weights to transitions. This is an equivalent representation of a WTS over  $(\{\rightarrow\}, \Sigma)$ .

■ **Table 1** Transitions of the weighted word automaton  $\mathcal{B}_k$ .

$\delta \xrightarrow{(i,a)} \delta'$	if $i \notin \text{dom}(\delta)$ . Then, $\text{dom}(\delta') = \text{dom}(\delta) \cup \{i\}$ , $\delta'(j) = \delta(j)$ for all $j \in \text{dom}(\delta)$ , and $\delta'(i) = (f_\emptyset, q, a, f_\emptyset)$ for some $q \in Q$ . The weight of this transition is $1_S$ .
$\delta \xrightarrow{\text{Forget}_i} \delta'$	if $i \in \text{dom}(\delta)$ . Then $\delta'$ is the restriction of $\delta$ to $\text{dom}(\delta') = \text{dom}(\delta) \setminus \{i\}$ . The weight of this transition is $\text{wgt}(\delta(i))$ .
$\delta \xrightarrow{\text{Add}_{i,j}^\gamma} \delta'$	if $i, j \in \text{dom}(\delta)$ , $i \neq j$ , $\gamma \notin \text{dom}(f_{\text{out}}(i))$ and $\gamma \notin \text{dom}(f_{\text{in}}(j))$ . Then, $\text{dom}(\delta') = \text{dom}(\delta)$ , $\delta'(\ell) = \delta(\ell)$ for all $\ell \in \text{dom}(\delta) \setminus \{i, j\}$ , $\delta'(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i) \cup [\gamma \mapsto q(j)])$ , $\delta'(j) = (f_{\text{in}}(j) \cup [\gamma \mapsto q(i)], q(j), a(j), f_{\text{out}}(j))$ . The weight of this transition is $1_S$ .

► **Lemma 12.** *Given a WTS  $\mathcal{T}$  over  $(\Gamma, \Sigma)$ -graphs and  $k > 0$ , we can compute in polynomial time wrt.  $\mathcal{T}$ , a weighted word automaton  $\mathcal{B}_k$  which is equivalent to  $\mathcal{T}$  over graphs of path width at most  $k$ . That is, for all  $k$ -words  $\tau$  with  $\{\tau\} = (G, \emptyset)$ , we have  $\llbracket \mathcal{T} \rrbracket(G) = \llbracket \mathcal{B}_k \rrbracket(\tau)$ .*

**Proof.** Let  $\mathcal{T} = (Q, \Delta, \text{wgt})$  be a WTS over  $(\Gamma, \Sigma)$ -graphs. By adding tiles with weight  $0_S$ , we may assume wlog that  $\Delta$  contains all possible tiles. Fix  $k \geq 1$ .

A state of  $\mathcal{B}_k$  is a partial map  $\delta: [k] \rightarrow \Delta$ . When reading a  $k$ -word  $\tau$  with  $\{\tau\} = (G, \chi)$ , the automaton will guess a labelling  $\rho: V \rightarrow Q$  of vertices of  $G$  with states of  $\mathcal{T}$  and will reach a state  $\delta$  satisfying the following two conditions:

1.  $\text{dom}(\delta) = \text{dom}(\chi) \subseteq [k]$  is the set of active colors,
2. for each active color  $i \in \text{dom}(\chi)$ ,  $\delta(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i)) = \text{tile}_\rho(\chi(i))$  is the current  $\rho$ -tile at vertex  $\chi(i)$  in  $G$ .

The only initial state is the empty map  $\delta_\emptyset$  with  $\text{dom}(\delta_\emptyset) = \emptyset$ . This is also the only final state, which is reached on a  $k$ -word  $\tau$  if all colors have been forgotten:  $\{\tau\} = (G, \chi_\emptyset)$ .

Transitions of the word automaton  $\mathcal{B}_k$  are given in Table 1. As above, we write  $\delta(i) = (f_{\text{in}}(i), q(i), a(i), f_{\text{out}}(i))$  and  $\delta'(i) = (f'_{\text{in}}(i), q'(i), a'(i), f'_{\text{out}}(i))$ .

The number of partial maps from  $A$  to  $B$  is  $(1 + |B|)^{|A|}$ . Hence, the number of states of  $\mathcal{B}_k$  is  $(1 + |\Delta|)^{1+k}$ . In a tile  $(f_{\text{in}}, q, a, f_{\text{out}}) \in \Delta$ , both  $f_{\text{in}}$  and  $f_{\text{out}}$  can be seen as partial maps from  $\Gamma$  to  $Q$ . Hence,  $|\Delta| = (1 + |Q|)^{2|\Gamma|} \cdot |Q| \cdot |\Sigma|$ . Also,  $|\Omega_k| = (1 + k)(|\Sigma| + 1) + (1 + k)^2|\Gamma|$ . We deduce that, if  $\Sigma, \Gamma, k$  are fixed, the automaton  $\mathcal{B}_k$  can be constructed in polynomial time wrt. the given WTS  $\mathcal{T}$ . ◀

## 4.2 Bounded tree-width evaluation

We extend the efficient evaluation of WTS for graphs of bounded path-width to graphs of bounded tree-width, which is a larger class of graphs. For instance, nested words may have unbounded path-width but their tree-width is at most 2. As for path-width, tree-width can be defined via tree decompositions: instead of a sequence of subsets of vertices, we use a tree of subsets of vertices. Since we will use weighted tree automata to achieve the efficient evaluation over graphs of bounded tree-width, we define directly tree terms. These are similar to  $k$ -words, with an additional binary union  $\oplus$ .

**Tree terms (TTs).** form an algebra to define labeled graphs. With  $a \in \Sigma$ ,  $\gamma \in \Gamma$  and  $i, j \in [k] = \{0, 1, \dots, k\}$ , the syntax of  $k$ -TTs over  $(\Gamma, \Sigma)$  is given by

$$\tau ::= (i, a) \mid \text{Add}_{i,j}^\gamma \tau \mid \text{Forget}_i \tau \mid \tau \oplus \tau$$

Each  $k$ -TT represents a colored graph  $\{\tau\} = (G_\tau, \chi_\tau)$  where  $G_\tau$  is a  $(\Gamma, \Sigma)$ -labeled graph and  $\chi_\tau: [k] \rightarrow V$  is a partial injective function coloring some vertices of  $G_\tau$ . Colors in  $\text{dom}\chi_\tau$  are said to be active in  $\tau$ . The semantics is defined as for  $k$ -words: a leaf  $(i, a)$  creates a graph with a single  $a$ -labeled vertex with color  $i$ ,  $\text{Forget}_i$  removes color  $i$  from the domain of the color map, and  $\text{Add}_{i,j}^\alpha$  adds an  $\alpha$ -labeled edge between the vertices colored  $i$  and  $j$  (if such vertices exist). Formally, if  $\{\tau\} = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda, \chi)$  then

- $\{\text{Add}_{i,j}^\alpha \tau\} = (V, (E'_\gamma)_{\gamma \in \Gamma}, \lambda, \chi)$  with  $E'_\gamma = E_\gamma$  if  $\gamma \neq \alpha$  and

$$E'_\alpha = \begin{cases} E_\alpha & \text{if } \{i, j\} \notin \text{dom}(\chi) \\ E_\alpha \cup \{(\chi(i), \chi(j))\} & \text{otherwise.} \end{cases}$$

- $\{\text{Forget}_i \tau\} = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda, \chi')$  with  $\text{dom}(\chi') = \text{dom}(\chi) \setminus \{i\}$  and  $\chi'(j) = \chi(j)$  for all  $j \in \text{dom}(\chi')$ .

The main difference with  $k$ -words is  $\oplus$  which takes the union of the two graphs, merging vertices with the same colors, if any.

- Formally, consider  $\tau' \oplus \tau''$  with  $\{\tau'\} = (G', \chi') = (V', (E'_\gamma)_{\gamma \in \Gamma}, \lambda', \chi')$  and  $\{\tau''\} = (G'', \chi'') = (V'', (E''_\gamma)_{\gamma \in \Gamma}, \lambda'', \chi'')$ . Let  $I = \text{dom}(\chi') \cap \text{dom}(\chi'')$  be the set of colors that are defined in both graphs. Wlog, we may assume that  $V' \cap V'' = \chi'(I) = \chi''(I)$  and  $\chi'(i) = \chi''(i)$  for all  $i \in I$ , i.e., we may rename the vertices so that the shared colors define the shared vertices. The union  $\tau' \oplus \tau''$  is well-defined only if the shared vertices have the same labels:  $\lambda'(\chi'(i)) = \lambda''(\chi''(i))$  for all  $i \in I$ . Then,  $\{\tau' \oplus \tau''\} = (G' \cup G'', \chi' \cup \chi'') = (V, (E_\gamma)_{\gamma \in \Gamma}, \lambda, \chi)$  where  $V = V' \cup V''$ ,  $\lambda = \lambda' \cup \lambda''$ , and  $E_\gamma = E'_\gamma \cup E''_\gamma$  for all  $\gamma \in \Gamma$ .

The tree-width of a nonempty graph  $G$  is the least  $k \geq 1$  such that  $G = G_\tau$  for some  $k$ -TT  $\tau$ .

Trees have tree-width 1, and as a special case, words also have tree-width 1. Nested words have tree-width (at most) 2 [21]. They are words with an additional binary relation from pushes to matching pops, which are used to represent behaviours of pushdown automata. On the other end, grids as used for instance in Example 4, have unbounded tree-width. More precisely, an  $n \times n$  grid has tree-width  $n$ .

We will focus on a regular subset of terms which ensures that the semantics is well-defined and that the  $k$ -TTs do not contain redundant operations such as  $\text{Add}_{i,j}^\gamma \text{Add}_{i,j}^\gamma \tau$  or  $\text{Add}_{i,j}^\gamma \tau_1 \oplus \text{Add}_{i,j}^\gamma \tau_2$ . A  $k$ -TT is *well-formed* if the following are satisfied:

1. if  $\text{Forget}_i \tau'$  is a subterm of  $\tau$  then  $i$  is active in  $\tau'$ ,
2. if  $\text{Add}_{i,j}^\gamma \tau'$  is a subterm of  $\tau$  then  $i, j$  are active in  $\tau'$  and the edge  $\gamma$  was not already added in  $\tau'$  between  $\chi_{\tau'}(i)$  and  $\chi_{\tau'}(j)$ .
3. if  $\tau' \oplus \tau''$  is a subterm of  $\tau$  then for all  $i, j$  that are active in both  $\tau'$  and  $\tau''$ , the vertices  $\chi_{\tau'}(i)$  and  $\chi_{\tau''}(i)$  have the same label from  $\Sigma$ , and we do not already have a  $\gamma$ -edge both between  $(\chi_{\tau'}(i), \chi_{\tau'}(j))$  and  $(\chi_{\tau''}(i), \chi_{\tau''}(j))$ .

The problem  $k$ -TW-FVal is to compute  $\llbracket \mathcal{T} \rrbracket(G)$ , given a WTS  $\mathcal{T}$  and a  $(\Gamma, \Sigma)$ -graph  $G$  of tree-width at most  $k$ .

► **Theorem 13.** *The problem  $k$ -TW-FVal can be solved in linear time wrt. the input graph  $G$  and polynomial time wrt. the input WTS  $\mathcal{T}$ .*

**Proof.** The proof follows the same three steps as for Theorem 11 using tree terms instead of  $k$ -words and weighted tree automata instead of weighted word automata.

1. From the input graph  $G$ , which is assumed to be of tree-width at most  $k$ , we compute in linear time a tree decomposition using Bodlaender's algorithm [3]. Then, similarly to Lemma 10, we compute in linear time a well-formed  $k$ -TT  $\tau$  such that  $\{\tau\} = (G, \emptyset)$ . In particular,  $|\tau| = \mathcal{O}(|G|)$ .

■ **Algorithm 1** Evaluation algorithm for a weighted tree automaton  $\mathcal{B} = (Q, T, F, \text{wgt})$ .

---

```

1: function MAIN( $\tau$ : term): value from  $S$  ▷ Computes  $[[\mathcal{B}]](\tau)$ 
2:    $\text{val} \leftarrow \text{TREEEVAL}(\tau)$ ;  $x \leftarrow 0_{\mathbb{S}}$ 
3:   for all  $q \in F$  do  $x \leftarrow x + \text{val}[q]$  end for
4:   return  $x$ 
5: end function
6: function TREEEVAL( $\tau$ : term): array indexed by  $Q$  of values from  $S$ 
7:   ▷ TREEEVAL( $\tau$ )[ $q$ ] is the sum of the weights of the runs of  $\mathcal{B}$  on  $\tau$  reaching state  $q$ .
8:   match  $\tau$  with
9:     Leaf  $a$ :
10:    for all  $q \in Q$  do  $\text{val}[q] \leftarrow \text{wgt}(\perp, a, q)$  end for
11:    Unary  $a(\tau_1)$ :
12:     $\text{val}_1 \leftarrow \text{TREEEVAL}(\tau_1)$ 
13:    for all  $q \in Q$  do  $\text{val}[q] \leftarrow 0_{\mathbb{S}}$  end for
14:    for all  $(q_1, a, q) \in T$  do  $\text{val}[q] \leftarrow \text{val}[q] + \text{val}_1[q_1] \times \text{wgt}(q_1, a, q)$  end for
15:    Binary  $a(\tau_1, \tau_2)$ :
16:     $\text{val}_1 \leftarrow \text{TREEEVAL}(\tau_1)$ ;  $\text{val}_2 \leftarrow \text{TREEEVAL}(\tau_2)$ 
17:    for all  $q \in Q$  do  $\text{val}[q] \leftarrow 0_{\mathbb{S}}$  end for
18:    for all  $(q_1, q_2, a, q) \in T$  do
19:       $\text{val}[q] \leftarrow \text{val}[q] + \text{val}_1[q_1] \times \text{wgt}(q_1, q_2, a, q) \times \text{val}_2[q_2]$ 
20:    end for
21:  end match
22:  return  $\text{val}$ 
23: end function

```

---

2. Using Lemma 14 below, from the WTS  $\mathcal{T}$  we construct in polynomial time an equivalent weighted tree automaton  $\mathcal{B}_k$  on graphs of tree-width at most  $k$ :  $[[\mathcal{T}]](G) = [[\mathcal{B}_k]](\tau)$ .
3. We compute  $[[\mathcal{B}_k]](\tau)$  with Algorithm 1. The main complexity comes from the call TREEEVAL. Executing the body of this function (without the recursive calls) takes time  $\mathcal{O}(|\mathcal{B}_k|)$ . Hence, the overall time complexity of this evaluation is  $\mathcal{O}(|\tau| \cdot |\mathcal{B}_k|)$ . ◀

A weighted (binary) tree automaton over alphabet  $\Sigma$  is usually given as a tuple  $\mathcal{B} = (Q, T, F, \text{wgt})$  where  $F \subseteq Q$  is the subset of accepting states,  $T \subseteq (\{\perp\} \cup Q \cup Q^2) \times \Sigma \times Q$  defines the bottom-up transitions and  $\text{wgt}: T \rightarrow S$  gives weights to transitions. This is an equivalent representation of a WTS over  $(\{\nearrow, \nwarrow\}, \Sigma)$ .

► **Lemma 14.** *Given a WTS  $\mathcal{T}$  over  $(\Gamma, \Sigma)$ -graphs and  $k > 0$ , we can compute in polynomial time wrt.  $\mathcal{T}$ , a weighted tree automaton  $\mathcal{B}_k$  which is equivalent to  $\mathcal{T}$  over graphs of tree-width at most  $k$ . Here, equivalent means that for all well-formed  $k$ -TTs  $\tau$  with  $\{\tau\} = (G, \emptyset)$ , we have  $[[\mathcal{T}]](G) = [[\mathcal{B}_k]](\tau)$ .*

## 5 Discussions and conclusions

**Connections with CSP.** The quantitative versions of the constraint satisfaction problem (CSP) are closely related to the evaluation problem for weighted tiling systems and graphs. Classic (boolean) CSPs ask for the existence of a solution of a set of constraints, as non-deterministic automata ask for the existence of an accepting run. In the *valued*-CSP (see

e.g. [20]), weights (costs) are assigned to each constraint depending on how the constraint is fulfilled, these weights are summed over all constraints and the aim is to minimize this total cost. This corresponds to our evaluation problem in the **min-plus** tropical semiring.

The weighted counting CSP (weighted  $\#$ CSP) is defined similarly but uses a  $(+, \times)$ -semiring such as  $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \dots$ , (see e.g. [7, 8]). The cost of a solution is the product of the weights over all constraints and the value of the weighted  $\#$ CSP is the sum over all solutions. Counting CSP ( $\#$ CSP) is obtained with semiring **Natural** when functions in the language only take values 0 or 1, thus counting the number of solutions of the classic CSP.

One of the main problems in CSP is to determine conditions under which the problems are tractable (polynomial time). Feder and Vardi conjectured [16] that, depending on the constraint language  $\Gamma$ , problems in  $\text{CSP}(\Gamma)$  are either in **P** or **NP**-complete. The dichotomy conjecture extends to  $\#$ CSP( $\Gamma$ ), saying that such counting problems are either in **FP** or  $\#$ **P**-complete, see e.g. [6, 15]. In this paper, we show that for WTS, the evaluation problem is  $\#$ **P**-complete (Theorem 7).

Most often the *non-uniform* complexity is considered, meaning that the language (for us the WTS) is not part of the input and the complexity only depends on the instance (for us the input graph). One such structural restriction is when the constraint graph of the instance has bounded tree-width. This is indeed related to our efficient evaluation described in Section 4. Our approach is different though since we reduce WTS to weighted word/tree automata and obtain a complexity linear in the input graph.

As future work, we plan to investigate more closely the relationship between weighted  $\#$ CSP and the evaluation problem for WTS. In particular, it would be interesting to see whether results on approximate computation which are widely studied for quantitative CSP can be transferred to weighted tiling systems.

**On the generality of the model.** The model of WGA [10] additionally has occurrence constraints (boolean combinations of constraints of the form  $\#\text{tile} \geq n$ , where  $\text{tile} \in \Delta$  and  $n \in \mathbb{N}$ ). A run is valid only if the occurrence constraints are satisfied. We could allow these constraints as well, without compromising the complexity upper bounds. In fact, we can allow more expressive quantifier-free Presburger constraints on the tiles (e.g.,  $\#\text{tile}_1 + \#\text{tile}_2 = \#\text{tile}_3$ ). The NP machine witnessing the upper bounds can compute the Parikh vector of the tiles used in a guessed run, and check in polynomial time whether the constraints are satisfied.

**Variants.** The evaluation problem **Eval** is a function problem. The decision variants correspond to threshold languages such as, is the computed weight  $\{>, \geq, <, \leq, =, \neq\}$   $s$ ,  $s$  being a threshold. There are further variants depending on whether the threshold  $s$  is part of the input or is fixed. The complexity depend on the semiring as well as on the value of the threshold when it is fixed.

**Conclusion.** We have given tight complexity bounds for the evaluation problem for various semirings. Our complexity upper bounds allows weights to be given in binary for problems over  $(+, \times)$ -semirings. However for tropical semirings the weights are assumed to be given in unary. While our upper bounds hold for arbitrary graphs, lower bounds are given uniformly for pictures (grid graphs). Further if we assume that the input graph does not have unbounded grid as a minor (bounded tree-width), then we provide efficient evaluation algorithm.

## References

- 1 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012.
- 2 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In (*ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17, Sidney, Australia, November 2014. Springer. doi:10.1007/978-3-319-11936-6\_1.
- 3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, December 1996.
- 4 Benedikt Bollig and Paul Gastin. Non-sequential theory of distributed systems. *CoRR*, abs/1904.06942, 2019. URL: <http://arxiv.org/abs/1904.06942>.
- 5 Benedikt Bollig and Ingmar Meinecke. Weighted distributed systems and their logics. In *International Symposium on Logical Foundations of Computer Science, LFCSS 2007*, volume 4514 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2007.
- 6 Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. *Inf. Comput.*, 205(5):651–678, 2007.
- 7 Andrei A. Bulatov, Martin E. Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #csp. *J. Comput. Syst. Sci.*, 78(2):681–688, 2012.
- 8 Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints An Int. J.*, 21(2):115–144, 2016.
- 9 Aiswarya Cyriac. *Verification of communicating recursive programs via split-width. (Vérification de programmes récurifs et communicants via split-width)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: <https://tel.archives-ouvertes.fr/tel-01015561>.
- 10 Manfred Droste and Stefan Dück. Weighted automata and logics on graphs. In *Mathematical Foundations of Computer Science (MFCS'15)*, volume 9234 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2015.
- 11 Manfred Droste and Paul Gastin. The kleene-schützenberger theorem for formal power series in partially commuting variables. *Inf. Comput.*, 153(1):47–80, 1999.
- 12 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Springer Berlin Heidelberg, 2009.
- 13 Manfred Droste, Christian Pech, and Heiko Vogler. A Kleene theorem for weighted tree automata. *Theory Comput. Syst.*, 38(1):1–38, 2005.
- 14 Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theor. Comput. Sci.*, 366(3):228–247, 2006.
- 15 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. The complexity of weighted boolean #csp. *SIAM J. Comput.*, 38(5):1970–1986, 2009.
- 16 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.
- 17 Ina Fichtner. Weighted picture automata and weighted logics. *Theory Comput. Syst.*, 48(1):48–78, 2011.
- 18 Paul Gastin and Benjamin Monmege. Adding pebbles to weighted automata: Easy specification & efficient evaluation. *Theoretical Computer Science*, 534:24–44, May 2014.
- 19 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. doi:10.1016/0022-0000(88)90039-6.
- 20 Andrei A. Krokhin and Stanislav Zivny. The complexity of valued csps. In Andrei A. Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 233–266. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

- 21 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011.
- 22 Christian Mathissen. Weighted logics for nested words and algebraic formal power series. *Logical Methods in Computer Science*, 6(1), February 2010.
- 23 Ingmar Meinecke. Weighted logics for traces. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *First International Computer Science Symposium in Russia, CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2006.
- 24 Wolfgang Thomas. On logics, tilings, and automata. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo, editors, *Automata, Languages and Programming*, pages 441–454, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 25 L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.



# Process Symmetry in Probabilistic Transducers

Shaul Almagor 

Computer Science Department, Technion, Haifa, Israel  
shaull@cs.technion.ac.il

---

## Abstract

Model checking is the process of deciding whether a system satisfies a given specification. Often, when the setting comprises multiple processes, the specifications are over sets of input and output signals that correspond to individual processes. Then, many of the properties one wishes to specify are symmetric with respect to the processes identities. In this work, we consider the problem of deciding whether the given system exhibits symmetry with respect to the processes' identities. When the system is symmetric, this gives insight into the behaviour of the system, as well as allows the designer to use only representative specifications, instead of iterating over all possible process identities.

Specifically, we consider probabilistic systems, and we propose several variants of symmetry. We start with precise symmetry, in which, given a permutation  $\pi$ , the system maintains the exact distribution of permuted outputs, given a permuted inputs. We proceed to study approximate versions of symmetry, including symmetry induced by small  $L_\infty$  norm, variants of Parikh-image based symmetry, and qualitative symmetry. For each type of symmetry, we consider the problem of deciding whether a given system exhibits this type of symmetry.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Verification by model checking; Theory of computation  $\rightarrow$  Concurrency; Theory of computation  $\rightarrow$  Abstraction

**Keywords and phrases** Symmetry, Probabilistic Transducers, Model Checking, Permutations

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.35

**Funding** *Shaul Almagor*: Supported by a European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.

**Acknowledgements** The author thanks Gal Vardi for discussions on the motivation for this work.

## 1 Introduction

A fundamental approach to automatic verification is *model checking* [4], where we are given a system and a specification, and we check whether all possible behaviours of the system satisfy the specification. In model checking of *reactive* systems, the specification is over sets of inputs  $I$  and outputs  $O$ , and the system is an  $I/O$  transducer, which takes sequences of inputs in  $2^I$ , and responds with an output in  $2^O$ . Then, model checking amounts to deciding whether for every input sequence, the matching output sequence generated by the transducer, satisfies the specification.

In practice, and especially in verification of concurrent systems, the input and output sets have some correspondence. For example, in an arbiter for  $k$  processes, the inputs are typically  $I = \{i_1, \dots, i_k\}$ , where  $i_j$  is interpreted as “a request was generated by Process  $j$ ”, and the outputs are  $O = \{o_1, \dots, o_k\}$ , where  $o_j$  is interpreted as “Process  $j$  was granted access”. In such cases, specification often end up having symmetric repetitions of a similar pattern. For example, we may wish to specify that in our arbiter, if Process  $j_1$  generated a request before Process  $j_2$ , then a grant for  $j_1$  should be given before a grant for  $j_2$ . However, in order to specify this in e.g., LTL (Linear Temporal Logic), we would have to explicitly write this statement for every pair of processes  $j_1, j_2$ . In the worst case, this could entail a blowup of  $k!$  in the size of the formula, which incurs a further exponential blowup during model-checking algorithms.



© Shaul Almagor;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 35; pp. 35:1–35:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This drawback, however, vanishes when we consider a *symmetric* system: intuitively, a system is symmetric if permuting the input signals generates an output sequence of similarly permuted outputs. If a system satisfies this property, then it is enough to check whether it satisfies a representative specification. Indeed, any permutation of the processes is guaranteed to be equivalently satisfied.

Unfortunately, deterministic systems are unlikely to be completely symmetric, unless they are very naive (e.g., no grants are ever given). Indeed, tie-breaking in deterministic systems has an inherent asymmetry to it. In *probabilistic* systems, however, no asymmetry is needed to break ties – one can randomly choose a result.

In this paper, we consider several notions of symmetry for probabilistic transducers, and their corresponding decision procedures. We start with the most restrictive version of symmetry, in which a transducer  $\mathcal{T}$  is symmetric under a permutation if the distribution of outputs that are generated for an input sequence  $x$  is identical to the distribution of permuted outputs for the permuted input sequence (Section 3). We show that deciding whether a transducer is symmetric under a given permutation is decidable in polynomial time, and use basic results in group theory to give a similar result for deciding whether a transducer is symmetric under all permutations in a permutation group.

We then proceed to study approximate notions of symmetry, in order to capture cases where a system is not fully symmetric, but still may exhibit some symmetrical properties. On the negative side, using results on probabilistic automata, we show that an  $L_\infty$  approximation variant of symmetry results in undecidability. On the positive side, we study two variants of symmetry that only take into account the Parikh image of the output signals, and we are able to use results on probabilistic automata with rewards to obtain efficient decidability of symmetry for these variants (Section 4).

Finally, we study a qualitative version of symmetry, which offers a coarse “nondeterministic” approximation of symmetry (Section 5). We show that deciding whether a system is qualitatively symmetric is PSPACE complete.

The notion of symmetry is not only appealing for symmetry reductions in specification, but also as a standalone feature for the *explainability* of model checking: standard model-checking algorithms can output a counterexample whenever a system does not satisfy its specification. This gives the designer insight as to what is wrong with either the system or the specification. On the other hand, when the result of model checking is that a system does satisfy its specification, no additional information is typically given. While this is “good news”, a designer often wants some information as to “why” the system is correct. In particular, the designer may be concerned that the specifications were too easy to satisfy (e.g., in vacuous specifications [1]). In this case, symmetry provides some information. Indeed, symmetry can be easily witnessed (as we show in Remark 4), so the designer can be convinced that any weakness of the specification, or any flaw of the system, is not biased toward a specific process, and will arise regardless of a specific order of processes. In addition, it shows that if the system satisfies e.g., liveness properties, then it satisfies them with the same “good event intervals” regardless of process identities.

## Related work

Process symmetry [3, 8, 6, 12] and more general symmetry reductions [16, 17, 19] have been studied since the 90’s, typically in the context of alleviating the state-explosion problem. Symmetry can either be specified by the designer or user [13,24,25], or detected automatically [15,16,32].

A close approach to our work here is [12], where the problem of detecting process symmetries is studied. There, however, parametrized deterministic systems are studied, which shift the focus to the pattern of given symmetries (rather than our fixed-length permutations), and does not concern probabilities.

Symmetry in the probabilistic setting was studied in [11, 5], where model checking of probabilistic systems exploits known symmetries to avoid a state blowup by considering a quotient of the system under the symmetry.

We remark that the works above typically focus on exact symmetries, and use them to reduce the state space, whereas the focus of this paper is to decide whether a symmetry exists, for various types of (not necessarily exact) symmetries, and to use the symmetry to avoid blowup in the specification, as well as to give the user insight regarding the correctness of the system.

Due to lack of space, some proofs appear in the appendix.

## 2 Preliminaries

### Probabilities and Distributions

Consider a finite set  $S$ . A *distribution* over  $S$  is a function  $\mu : S \rightarrow [0, 1]$  such that  $\sum_{s \in S} \mu(s) = 1$ . We denote the space of all distributions over  $S$  by  $\Delta(S)$ . Given a distribution  $\mu$ , an *event* is a subset<sup>1</sup>  $E \subseteq S$ , and its *probability* under  $\mu$  is  $\Pr(E) = \sum_{e \in E} \mu(e)$ . For an

element  $s \in S$ , the *Dirac distribution*  $\mathbf{1}[s]$  is given by  $\mathbf{1}[s](r) = \begin{cases} 1 & r = s, \\ 0 & r \neq s. \end{cases}$  The *support* of a distribution  $\mu$  is  $\text{Supp}(\mu) = \{s \in S : \mu(s) > 0\}$ .

Given sets  $S_1, \dots, S_n$  and distributions  $\mu_1, \dots, \mu_n$  such that  $\mu_i \in \Delta_i$  for every  $1 \leq i \leq n$ , a natural *product distribution*  $\mu$  is induced on the product space  $S_1 \times \dots \times S_n$  where  $\mu(s_1, \dots, s_n) = \prod_{i=1}^n \mu_i(s_i)$ .

### Probabilistic Transducers and Automata

Consider two finite sets  $I$  and  $O$  of input and output signals, respectively. An *I/O probabilistic transducer* (henceforth just *transducer*) is  $\mathcal{T} = \langle I, O, S, s_0, \delta, \ell \rangle$  where  $S$  is a finite set of states,  $s_0$  is an initial state,  $\delta : S \times 2^I \rightarrow \Delta(S)$  is a transition function, assigning to each (state, letter) pair a distribution of successor states, and  $\ell : S \rightarrow 2^O$  is a labelling function.

For a word  $x = \mathbf{i}_1 \cdot \mathbf{i}_2 \cdots \mathbf{i}_n \in (2^I)^+$ , a *run* of  $\mathcal{T}$  on  $x$  is a sequence  $\rho = q_0, q_1, \dots, q_n$  where  $q_0 = s_0$ , and the *probability* of the run  $\rho$  is  $\prod_{j=0}^{n-1} \delta(q_j, \mathbf{i}_{j+1})(q_{j+1})$ . Note that indeed this induces a probability measure  $\mu$  on  $\{s_0\} \times S^n$  via the product distribution.

A run  $\rho$  is *proper* if  $\rho \in \text{Supp}(\mu)$ . That is, if it has positive probability. We denote the space of proper runs by  $\text{runs}(\mathcal{T}, x)$ . In the following, we usually refer only to proper runs, and we omit the term “proper” when it is clear from context. We extend the labelling function  $\ell$  to runs by  $\ell(\rho) = \ell(q_1) \cdot \ell(q_2) \cdots \ell(q_n)$ . Observe that we ignore the labelling of the initial state, and only consider nonempty words, to avoid edge cases.

For  $x \in (2^I)^+$  and  $y \in (2^O)^+$  such that  $|x| = |y|$ , we denote by  $\mathcal{T}(x) = y$  the event  $\{\rho \in \text{runs}(\mathcal{T}, x) : \ell(\rho) = y\}$ . Thus,  $\Pr(\mathcal{T}(x) = y)$  is the probability that the output generated by  $\mathcal{T}$  on input  $x$  is exactly  $y$ . We denote by  $x \otimes y \in (2^{I \cup O})^\omega$  the combined word  $(\mathbf{i}_1 \cup \mathbf{o}_1) \cdot (\mathbf{i}_2 \cup \mathbf{o}_2) \cdots (\mathbf{i}_n \cup \mathbf{o}_n)$ .

<sup>1</sup> In general  $E$  needs to be a *measurable subset*, but since we only consider finite sets, any subset is measurable.

The sets  $I$  and  $O$  are called *corresponding signals* if  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ . Intuitively, for  $1 \leq j \leq k$  we think of  $i_j$  as a request generated by a process  $j$ , and of  $o_j$  as a corresponding grant generated by the system.

A *probabilistic automaton (PA)* is  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  where  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet,  $\delta : Q \times \Sigma \rightarrow \Delta(Q)$  is a probabilistic transition function,  $q_0 \in Q$  is an initial state, and  $F \subseteq Q$  is a set of accepting states. Similarly to transducers, an input word  $x \in \Sigma^*$  induces a probability measure on the set  $\text{runs}(\mathcal{A}, x)$  of runs of  $\mathcal{A}$  on  $x$ . Then, we denote by  $\mathcal{A}(x)$  the probability that a run of  $\mathcal{A}$  on  $x$  is accepted, i.e. ends in a state in  $F$ .

### Permutations

We assume familiarity with basic notions in group theory (see e.g. [2]). A *permutation* of the set  $[k] = \{1, \dots, k\}$  is a bijection  $\pi : [k] \rightarrow [k]$ . A standard representation of permutations is by a *cycle decomposition*, where, for example, the cycle  $(1\ 2\ 7)$  represents the permutation  $\pi$  where  $\pi(1) = 2, \pi(2) = 7, \pi(7) = 1$ , and for all other elements we have  $\pi(j) = j$ . The set of all permutations on  $[k]$ , equipped with the functional composition operator  $\circ$  forms the *symmetric group*  $\mathcal{S}_k$ . Any subgroup of  $\mathcal{S}_k$  is referred to as a *permutation group*. A *generating set* of a permutation group  $G$  is a finite set  $X = \{\pi_1, \dots, \pi_m\}$  such that every permutation  $\tau \in G$  can be expressed as a composition of the elements in  $X$ . For such a set  $X$ , we denote the group generated by it by  $\langle X \rangle$ . It is well known that  $\{(1\ 2), (1\ 2\ \dots\ k)\}$  is a generating set of  $\mathcal{S}_k$  (see e.g., [2]).

Consider corresponding signals  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ , and let  $\pi \in \mathcal{S}_k$ . For a letter  $\mathbf{i} = \{i_{j_1}, \dots, i_{j_m}\} \in 2^I$ , we define  $\pi(\mathbf{i}) = \{i_{\pi(j_1)}, \dots, i_{\pi(j_m)}\}$ . That is,  $\pi$  permutes the signals given in  $\mathbf{i}$ .<sup>2</sup> Then, for a word  $x = \mathbf{i}_1 \cdot \mathbf{i}_2 \cdots \mathbf{i}_n \in (2^I)^+$ , we define  $\pi(x) = \pi(\mathbf{i}_1) \cdot \pi(\mathbf{i}_2) \cdots \pi(\mathbf{i}_n)$ . Similar definitions hold for  $O$ . Unless explicitly stated otherwise, we henceforth assume  $I$  and  $O$  are corresponding signals.

## 3 Symmetric Probabilistic Transducers

Let  $\mathcal{T} = \langle I, O, S, s_0, \delta, \ell \rangle$  be an  $I/O$  transducer over  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ , and let  $\pi \in \mathcal{S}_k$ . We say that  $\mathcal{T}$  is  $\pi$ -symmetric if for every  $x \in (2^I)^+$  and  $y \in (2^O)^+$  it holds that  $\Pr(\mathcal{T}(x) = y) = \Pr(\mathcal{T}(\pi(x)) = \pi(y))$ . That is,  $\mathcal{T}$  is  $\pi$ -symmetric if whenever we permute the input by  $\pi$ , the resulting distribution on outputs is permuted by  $\pi$  as well.

► **Example 1.** Consider a Round-Robin arbiter over three processes, as depicted in Figure 1. At each state, the arbiter looks for a request from a single processor  $j$ , and grants it if it is on, then moves to a state corresponding to process  $j + 1 \pmod{3}$ . Observe that this is a deterministic transducer, except that the initial state is unspecified.

Consider the case where we let the state marked 001 be initial, which corresponds to letting the first process start. In this case, the transducer is not  $\pi$ -symmetric for  $\pi = (1\ 2\ 3)$ . Indeed, the input word 100 will generate output 100, but its permutation  $\pi(100) = 010$  generates output  $000 \neq \pi(100)$ .

However, if we introduce a probabilistic initial state, that chooses each state of 100, 010, 001 as the next state, each with probability  $\frac{1}{3}$ , the transducer becomes  $\pi$ -symmetric for any  $\pi \in \mathcal{S}_3$ . ◀

<sup>2</sup> Formally, we would actually need  $I$  to be an ordered set. However, the order will be implied by the naming convention, so we let  $I$  be a set.



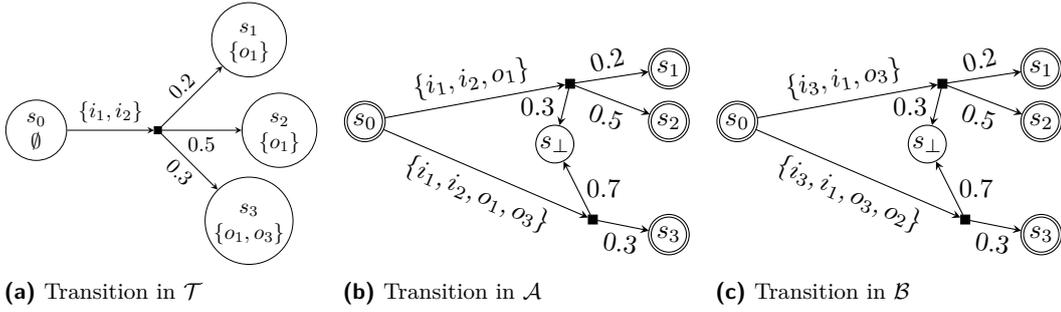
### 35:6 Process Symmetry in Probabilistic Transducers

Consider an  $I/O$  transducer  $\mathcal{T} = \langle I, O, S, s_0, \delta, \ell \rangle$  over  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ , and let  $\pi \in \mathcal{S}_k$ . We construct from  $\mathcal{T}$  two PAs  $\mathcal{A}$  and  $\mathcal{B}$ . Intuitively,  $\mathcal{A}$  mimics the behaviour of  $\mathcal{T}$ , by reading words over  $2^{I \cup O}$ , and accepting a word  $w \in (2^{I \cup O})^+$  with probability  $\mu$  iff  $\mathcal{T}$ , when reading the inputs that appear in  $w$ , generates the outputs that appear in  $w$  with probability  $\mu$ . The PA  $\mathcal{B}$  works exactly like  $\mathcal{A}$ , but permutes both the inputs and outputs by  $\pi$ .

Formally,  $\mathcal{A} = \langle S \cup \{q_\perp\}, 2^{I \cup O}, \eta, s_0, S \rangle$  and  $\mathcal{B} = \langle S \cup \{q_\perp\}, 2^{I \cup O}, \zeta, s_0, S \rangle$  where  $q_\perp$  is a new state, and the transition functions are defined as follows. Let  $q \in S$  and  $\sigma = \mathbf{i} \cup \mathbf{o}$  with  $\mathbf{i} \in 2^I$  and  $\mathbf{o} \in 2^O$ , and let  $V_p = \sum_{p \in S, \ell(p)=\mathbf{o}} \delta(q, \mathbf{i})(p)$  be the probability assigned by  $\mathcal{T}$  to seeing a state labelled  $\mathbf{o}$  after reading  $\mathbf{i}$  in state  $q$ , then  $\eta(q, \sigma) \in \Delta(S \cup \{q_\perp\})$  is the following distribution:

$$\eta(q, \sigma)(p) = \begin{cases} \delta(q, \mathbf{i})(p) & \text{if } p \in S \text{ and } \ell(p) = \mathbf{o} \\ 0 & \text{if } p \in S \text{ and } \ell(p) \neq \mathbf{o} \\ 1 - V_p & \text{if } p = q_\perp \end{cases}$$

In addition,  $\eta(q_\perp, \sigma)(q_\perp) = 1$  (so  $q_\perp$  is a rejecting sink). We demonstrate the construction of  $\mathcal{A}$  in Figures 2a and 2b.



**Figure 2** A transition in a transducer  $\mathcal{T}$  over  $I = \{i_1, i_2, i_3\}$  and  $O = \{o_1, o_2, o_3\}$ , and the corresponding transitions in  $\mathcal{A}$  and  $\mathcal{B}$ , under the permutation  $\pi = (1\ 2\ 3)$ . Observe that the transition in  $\mathcal{B}$  corresponds to the inverse permutation,  $\pi^{-1} = (3\ 2\ 1)$ , so that e.g.,  $\pi(\{i_3, i_1\}) = \{i_1, i_2\}$ .

The construction of  $\mathcal{B}$  is similar, but accounts for the permutation  $\pi$ . Let  $q \in S$  and  $\sigma = \mathbf{i} \cup \mathbf{o}$  with  $\mathbf{i} \in 2^I$  and  $\mathbf{o} \in 2^O$ , and let  $U_p = \sum_{p \in S, \ell(p)=\pi(\mathbf{o})} \delta(q, \pi(\mathbf{i}))(p)$  be the probability assigned by  $\mathcal{T}$  to seeing a state labelled  $\pi(\mathbf{o})$  after reading  $\pi(\mathbf{i})$  in state  $q$ , then  $\zeta(q, \sigma) \in \Delta(S \cup \{q_\perp\})$  is the following distribution:

$$\zeta(q, \sigma)(p) = \begin{cases} \delta(q, \pi(\mathbf{i}))(p) & \text{if } p \in S \text{ and } \ell(p) = \pi(\mathbf{o}) \\ 0 & \text{if } p \in S \text{ and } \ell(p) \neq \pi(\mathbf{o}) \\ 1 - U_p & \text{if } p = q_\perp \end{cases}$$

In addition,  $\zeta(q_\perp, \sigma)(q_\perp) = 1$  (so  $q_\perp$  is a rejecting sink). We demonstrate the construction of  $\mathcal{B}$  in Figures 2a and 2c.

Consider words  $x \in (2^I)^+$  and  $y \in (2^O)^+$ . Since  $q_\perp$  is the only rejecting state in both  $\mathcal{A}$  and  $\mathcal{B}$ , then by construction it is easy to see that  $\mathcal{A}(x \otimes y) = \Pr(\mathcal{T}(x) = y)$  and  $\mathcal{B}(x \otimes y) = \Pr(\mathcal{T}(\pi(x)) = \pi(y))$ . Thus, we have that  $\mathcal{A}$  and  $\mathcal{B}$  are equivalent iff  $\mathcal{T}$  is  $\pi$ -symmetric, and since equivalence can be decided in polynomial time, we are done.  $\blacktriangleleft$

Combining Theorem 5 with Corollary 3, we have the following.

► **Corollary 6.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a finite set of generators  $X = \{\pi_1, \dots, \pi_m\}$ , whether  $\mathcal{T}$  is  $\langle X \rangle$ -symmetric, is solvable in polynomial time.*

In particular, since the symmetric group  $\mathcal{S}_k$  is generated by two permutations  $\{(1\ 2), (1\ 2 \dots k)\}$ , we have the following.

► **Corollary 7.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$ , whether  $\mathcal{T}$  is  $\mathcal{S}_k$ -symmetric, is solvable in polynomial time.*

## 4 Approximate Symmetry

While aspiring to obtain symmetric systems is noble, in practice exact symmetry may be too strong a requirement, for example if the source of randomness supplies binary bits, and one needs e.g.,  $\frac{1}{3}$  probability, then only an approximate probability can be used. Thus, it is reasonable to seek approximate notions of symmetry.

### 4.1 $L_\infty$ Symmetry

The most straightforward approach toward approximate symmetry in probabilistic transducers is induced by the  $L_\infty$  norm, as follows. Let  $\mathcal{T}$  be an I/O-transducer, let  $\pi \in \mathcal{S}_k$ , and let  $\epsilon > 0$ . We say that  $\mathcal{T}$  is  $(\epsilon, \pi)$ -symmetric if  $|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = \pi(y))| \leq \epsilon$  for every  $x \in (2^I)^+$  and for every  $y \in (2^O)^+$ . That is, permuting the inputs by  $\pi$  perturbs the output distribution by at most  $\epsilon$ .

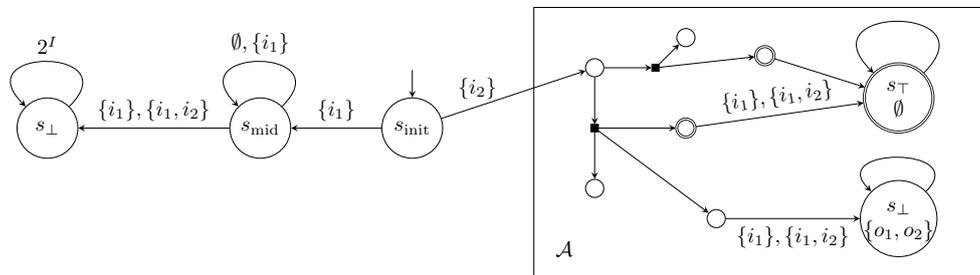
Unfortunately, as we now show, approximate symmetry is undecidable.

► **Theorem 8.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  a permutation  $\pi \in \mathcal{S}_k$  and  $\epsilon > 0$ , whether  $\mathcal{T}$  is  $(\epsilon, \pi)$ -symmetric, is undecidable.*

**Proof.** The *emptiness problem* for PA is to decide, given a PA  $\mathcal{A}$  over  $\Sigma$  and a threshold  $\lambda \in [0, 1]$ , whether there exists a word  $w \in \Sigma^*$  such that  $\mathcal{A}(w) > \lambda$ . This problem is known to be undecidable [14, 13, 7].

We show that approximate symmetry is undecidable via a reduction from a restriction of the emptiness problem (or rather the complement thereof), where the given PA is over the alphabet  $\{0, 1\}$ . The problem remains undecidable under this restriction, as we can encode any larger alphabet  $\Gamma$  using fixed-length sequences in  $\{0, 1\}^d$ , such that while reading the  $d$  symbols that compose a single letter in  $\Gamma$ , the states are not accepting (and hence we do not introduce a word whose acceptance probability is above  $\lambda$ ).

We start with an intuitive description of the reduction, depicted in Figure 3.



■ **Figure 3** The transducer constructed from a PA. The black squares denote probabilistic branching.

Consider a PA  $\mathcal{A}$  over the alphabet  $\Sigma = \{0, 1\}$ . We construct a transducer  $\mathcal{T}$  over  $I = \{i_1, i_2\}$  and  $O = \{o_1, o_2\}$  which has two components. Initially, if  $\mathcal{T}$  sees the input  $\{i_2\}$ , it moves to a component which mimics  $\mathcal{A}$  using the alphabet  $\{\emptyset, \{i_2\}\}$  instead of  $\{0, 1\}$ . At

this stage, all the states are marked with the output  $\{o_1, o_2\}$ . If at any point the input signal  $i_1$  is given, i.e. the letter  $\{i_1\}$  or  $\{i_1, i_2\}$ , then  $\mathcal{T}$  proceeds to a state labelled  $\{o_1, o_2\}$  from non-accepting states of  $\mathcal{A}$ , and to a state labelled  $\emptyset$  from accepting states. Thus, a word of the form  $\{i_2\} \cdot x \cdot \{\{i_1\}, \{i_1, i_2\}\}^*$  with  $x \in \{\emptyset, \{i_2\}\}^n$  would yield an output of the form  $\emptyset^{n+1} \cdot \emptyset^*$  with probability  $\mathcal{A}(x)$  and of the form  $\emptyset^{n+1} \cdot \{o_1, o_2\}^*$  with probability  $1 - \mathcal{A}(x)$ . Observe that both output possibilities are invariant under the permutation (1 2).

If, initially,  $\mathcal{T}$  sees the input  $\{i_1\}$ , it moves to a state labelled  $\emptyset$ , which loops as long as  $\{i_1\}$  or  $\emptyset$  are seen. Then, if  $\{i_2\}$  or  $\{i_1, i_2\}$  is seen, it moves to a sink labelled  $\{o_1, o_2\}$ . Essentially, this component mimics the output sequence of a rejecting run of  $\mathcal{A}$  in the first component, under the permutation (1 2). Hence, taking  $\epsilon = \lambda$ , we have that  $\mathcal{T}$  is  $(\epsilon, (1\ 2))$ -symmetric iff there does not exist a word  $x$  such that  $\mathcal{A}(x) > \lambda$ .

We proceed to give the precise reduction. Consider a PA  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  with  $\Sigma = \{0, 1\}$ , we construct an  $I/O$  transducer  $\mathcal{T} = \langle I, O, S, s_{\text{init}}, \eta, \ell \rangle$  as follows. The states of  $\mathcal{T}$  are  $S = Q \cup \{s_{\text{mid}}, s_{\text{init}}, s_{\top}, s_{\perp}\}$ , where  $s_{\perp} \notin Q$ , and the input and output sets are  $I = \{i_1, i_2\}$  and  $O = \{o_1, o_2\}$ . The labelling function is given by  $\ell(q) = \emptyset$  for all  $q \in Q$ ,  $\ell(s_{\perp}) = O = \{o_1, o_2\}$ , and  $\ell(s_{\text{init}}) = \ell(s_{\text{mid}}) = \{\emptyset\}$ . The transition function, as depicted in Figure 3, is defined as follows.

First, for every  $q \in Q$  and  $\mathbf{i} \in \{\emptyset, \{i_2\}\}$ , we have  $\eta(q, \mathbf{i}) = \delta(q, \mathbf{i})$ , where we identify  $\{\emptyset, \{i_2\}\}$  with  $\{0, 1\}$  in an arbitrary bijective manner. Next, if  $q \in F$ , then  $\eta(q, \{i_1\}) = \eta(q, \{i_1, i_2\}) = \mathbf{1}[s_{\top}]$ , and if  $q \notin F$  then  $\eta(q, \{i_1\}) = \eta(q, \{i_1, i_2\}) = \mathbf{1}[s_{\perp}]$ . The remaining transitions are

$$\begin{aligned} \eta(s_{\text{init}}, \{i_1\}) &= \mathbf{1}[s_{\text{mid}}], & \eta(s_{\text{mid}}, \emptyset) &= \eta(s_{\text{mid}}, \{i_1\}) = \mathbf{1}[s_{\text{mid}}], \\ \eta(s_{\text{init}}, \{i_2\}) &= \mathbf{1}[q_0], & \eta(s_{\text{mid}}, \{i_2\}) &= \eta(s_{\text{mid}}, \{i_1, i_2\}) = \mathbf{1}[s_{\perp}], \\ \eta(s_{\text{init}}, \emptyset) &= \eta(s_{\text{init}}, \{i_1, i_2\}) = \mathbf{1}[s_{\perp}], \end{aligned}$$

and for every  $\mathbf{i} \in 2^I$  we have  $\eta(s_{\perp}, \mathbf{i}) = \mathbf{1}[s_{\perp}]$  and  $\eta(s_{\top}, \mathbf{i}) = \mathbf{1}[s_{\top}]$ .

Let  $\pi = (1\ 2)$  and  $\epsilon = \lambda$ . Keeping our identification of  $\{\emptyset, \{i_2\}\}$  with  $\{0, 1\}$ , we claim that there exists a word  $x' \in \{\emptyset, \{i_2\}\}^*$  such that  $\mathcal{A}(x') > \lambda$  iff there exists words  $x \in (2^I)^+$  and  $y \in (2^O)^+$  such that  $|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = \pi(y))| > \epsilon$  (i.e.  $\mathcal{T}$  is not  $(\epsilon, \pi)$ -symmetric). Observe that  $\ell$  assigns only the labels  $\emptyset$  and  $\{o_1, o_2\}$ , both of which are invariant under  $\pi$ . Thus, the latter condition becomes

$$|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| > \epsilon. \quad (1)$$

We now turn to prove correctness. For the first direction, let  $x' \in \{\emptyset, \{i_2\}\}^*$  such that  $\mathcal{A}(x') > \lambda$ , and consider the word  $x = \{i_2\} \cdot x' \cdot \{i_1, i_2\}$ . By the construction of  $\mathcal{T}$ , after seeing  $\{i_2\}$ , there is only a single run of  $\mathcal{T}$  which proceeds to  $q_0$ . From there,  $\mathcal{T}$  mimics the behaviour of  $\mathcal{A}$  on  $x'$ . Thus, after reading  $x'$ , the distribution of states has probability  $\mathcal{A}(x)$  for states in  $F$ , and probability  $1 - \mathcal{A}(x)$  in states in  $Q \setminus F$ . Note that up until then, only the label  $\emptyset$  is seen, so the distribution of outputs is  $\mathbf{1}[\emptyset^{|x'|+1}]$ . Then, after reading  $\{i_1, i_2\}$ , the distribution of outputs give probability  $\mathcal{A}(x)$  to  $\emptyset^{|x'|+2}$ , and  $1 - \mathcal{A}(x)$  to  $\emptyset^{|x'|+1} \cdot \{o_1, o_2\}$ .

Now consider  $\pi(x) = \{i_1\} \cdot \pi(x') \cdot \{i_1, i_2\}$ . Upon reading  $\{i_1\}$ , the single run of  $\mathcal{T}$  arrives at  $s_{\text{mid}}$ . Then, since  $x' \in \{\emptyset, \{i_2\}\}^*$ , we have that  $\pi(x') \in \{\emptyset, \{i_1\}\}^*$ , so the run of  $\mathcal{T}$  stays in  $s_{\text{mid}}$ . Finally, reading  $\{i_1, i_2\}$ , the run moves to  $s_{\perp}$ . Therefore  $\mathcal{T}(x)$  gives probability 1 to the output  $\emptyset^{|x'|+1} \{o_1, o_2\}$ . Thus, for the output  $y = \emptyset^{|x'|+2}$ , we have that  $|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| = |\mathcal{A}(x) - 0| > \lambda = \epsilon$ , so  $\mathcal{T}$  is not  $(\epsilon, \pi)$ -symmetric.

For the converse direction, assume  $x, y$  are such that  $|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| > \epsilon$ . We start by eliminating candidates for such  $x$  and  $y$ . First, observe that if  $x$  starts with  $\emptyset$  or  $\{\beta_1, \emptyset_1\}$  (both of which are invariant under  $\pi$ ), we have  $\mathcal{T}(x)$  gives probability 1 to the output  $\ell(q_{\perp})^{|x|} = \{o_1, o_2\}^{|x|}$ , and so  $\mathcal{T}(x) = \mathcal{T}(\pi(x))$ , hence  $|\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| = 0$  for all  $y$ , so this case cannot occur.

Next, we claim that without loss of generality, we can assume  $x$  starts with  $\{i_2\}$ . Indeed, if  $x$  starts with  $\{i_1\}$ , then  $\pi(x)$  starts with  $\{i_2\}$ . Since  $\pi(\pi(x)) = x$ , we could start the argument with  $\pi(x)$ , while maintaining Equation (1).

Now, if  $x$  is of the form  $\{i_2\} \cdot \{\emptyset, \{i_2\}\}^n$ , then  $\mathcal{T}(x)$  gives probability 1 to the output  $\emptyset^{n+1}$ , but  $\pi(x)$  is now of the form  $\{i_1\} \cdot \{\emptyset, \{i_1\}\}^n$ , which also induces the same distribution, this case cannot occur as well.

It follows that  $x$  is of the form  $\{i_2\} \cdot x' \cdot \{\{i_1\}, \{i_1, i_2\}\} \cdot (2^I)^*$  where  $x' \in \{\emptyset, \{i_2\}\}^n$ . We claim that  $\mathcal{A}(x') > \lambda$ . Indeed, as we observed above,  $\mathcal{T}(x)$  gives probability  $\mathcal{A}(x')$  to the output  $\emptyset^{|x|}$  and probability  $1 - \mathcal{A}(x')$  to the output  $\emptyset^{|x'|+1} \cdot \{o_1, o_2\}^{|x|-|x'|-1}$ . However,  $\mathcal{T}(\pi(x))$  gives probability 1 to the output  $\emptyset^{|x'|+1} \cdot \{o_1, o_2\}^{|x|-|x'|-1}$ . Thus, there are only two possibilities for  $y$  in order for Equation (1) to hold: if  $y = \emptyset^{|x|}$ , we have

$$\lambda = \epsilon < |\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| = |\mathcal{A}(x') - 0| = \mathcal{A}(x')$$

and if  $y = \emptyset^{|x'|+1} \cdot \{o_1, o_2\}^{|x|-|x'|-1}$ , then

$$\lambda = \epsilon < |\Pr(\mathcal{T}(x) = y) - \Pr(\mathcal{T}(\pi(x)) = y)| = |1 - \mathcal{A}(x') - 1| = \mathcal{A}(x')$$

So in either case  $\mathcal{A}(x') > \lambda$ , and we are done.  $\blacktriangleleft$

A-priori, the fact that  $(\epsilon, \pi)$ -symmetry is undecidable does not mean that approximate symmetry for an entire permutation group is undecidable, nor that for fixed  $\epsilon$  the problem is undecidable. Unfortunately, however, the proof of Theorem 8 uses the permutation group  $\mathcal{S}_2$ , whose only nontrivial permutation is  $(1\ 2)$ . Moreover, the reduction uses the given threshold  $\lambda$  as is, by setting  $\lambda = \epsilon$ , and the emptiness problem is known to be undecidable even when  $\lambda$  is a fixed number in  $(0, 1)$ . Thus, we have the following.

► **Corollary 9.** *For every  $\epsilon \in (0, 1)$ , the problem of deciding, given an I/O transducer  $\mathcal{T}$  whether  $\mathcal{T}$  is  $(\epsilon, \pi)$ -symmetric for every  $\pi \in \mathcal{S}_k$ , is undecidable.*

► **Remark 10 (Composability).** While undecidability of  $(\epsilon, \pi)$ -symmetry is unfortunate, the reader may take solace in the fact that  $(\epsilon, \pi)$ -symmetry is anyway not preserved under composition. Indeed, if  $\mathcal{T}$  is  $(\epsilon, \pi)$ -symmetric and  $(\delta, \tau)$ -symmetric, it only guarantees that it is  $(\delta + \epsilon, \tau \cdot \pi)$ -symmetric. Thus, in order to ensure symmetry over a group, a sound method would have to take into account the *diameter* of the group. This, however, may lose completeness. Thus,  $(\epsilon, \pi)$ -symmetry is not a robust notion.

## 4.2 Parikh Symmetry

The notions of symmetry studied so far have a “letter-by-letter” flavour, where we compare the distribution of specific outputs for a given inputs. We now turn to study a different notion of symmetry, that abstracts away the order of the output symbols, and draws instead on the Parikh image of the computation.

Let  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ . For a word  $y = \mathbf{o}_1 \cdots \mathbf{o}_n \in 2^O$ , and  $1 \leq j \leq k$ , define  $\#(y, j) = |\{m : o_j \in \mathbf{o}_m\}|$  to be the number of occurrences of  $o_j$  in  $y$ . Then, we define the *Parikh image*<sup>3</sup> of  $y$  to be  $\mathfrak{P}(y) = (\#(y, 1), \dots, \#(y, k)) \in \mathbb{N}^k$ .

Given a permutation  $\pi$  and a vector  $\mathbf{a} = (a_1, \dots, a_k) \in \mathbb{N}^k$ , we define  $\pi(\mathbf{a}) = (a_{\pi^{-1}(1)}, \dots, a_{\pi^{-1}(k)})$ . Note that we use  $\pi^{-1}$  so that the following relation holds: if e.g.,  $\pi(1) = 3$ , then index 3 in  $\pi(\mathbf{a})$  contains  $a_1$ .

<sup>3</sup> Observe that this is not the standard Parikh image, in that it is the image with respect to signals in  $O$ , rather than to letters in  $2^O$ .

Consider an  $I/O$  transducer  $\mathcal{T}$  and a word  $x \in (2^I)^+$ . The outputs of  $\mathcal{T}$  on  $x$  induce a probability measure on (a finite subset of)  $\mathbb{N}^k$ , where for a vector  $\mathbf{a} \in \mathbb{N}^k$  we have  $\Pr(\mathfrak{P}(\mathcal{T}(x)) = \mathbf{a}) = \sum_{y: \mathfrak{P}(y) = \mathbf{a}} \Pr(\mathcal{T}(x) = y)$ . We can thus also consider the *expected* value of the Parikh image, given by  $\mathbb{E}[\mathfrak{P}(\mathcal{T}(x))] = \sum_y \Pr(\mathcal{T}(x) = y) \mathfrak{P}(y)$  (where the product is element-wise, so this is a vector in  $\mathbb{N}^k$ ).

Parikh images give rise to two measures of symmetry: given a permutation  $\pi$ , we say that  $\mathcal{T}$  is  $\pi$ -*Parikh distribution symmetric* if for every  $x \in (2^I)^+$  and every  $\mathbf{a} \in \mathbb{N}^k$  we have  $\Pr(\mathfrak{P}(\mathcal{T}(x)) = \mathbf{a}) = \Pr(\mathfrak{P}(\mathcal{T}(\pi(x))) = \pi(\mathbf{a}))$ . That is, every word  $x$  induces the same distribution of Parikh images as  $\pi(x)$  does for the permuted images. A weaker notion of symmetry uses expectation: we say that  $\mathcal{T}$  is  $\pi$ -*Parikh expected symmetric* if for every  $x \in (2^I)^+$  we have  $\mathbb{E}[\mathfrak{P}(\mathcal{T}(x))] = \pi(\mathbb{E}[\mathfrak{P}(\mathcal{T}(\pi(x)))])$

Note that Parikh-symmetry assumes the number of occurrences of a certain output signal is meaningful. This is relevant when the output signals measure e.g., number of grants for requests, but makes less sense when the outputs represent e.g., a choice between channels through which a message is routed.

Our algorithmic results about Parikh symmetry use a translation to *probabilistic reward automata* (PRA) [10, Section 5]. A PRA is a PA  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$  equipped with a *reward function*  $R : Q \rightarrow \{0, 1\}^k$  for some  $k \in \mathbb{N}$ .<sup>4</sup> The rewards are summed along a run, and the value of a word  $w \in \Sigma^*$ , denoted  $R(w)$ , is the expected reward, that is, the weighted sum of the rewards along all runs, weighted by their respective probabilities. We denote by  $\mathcal{A}(w)$  the distribution of reward vectors in  $\mathbb{N}^k$ , induced by the runs of  $\mathcal{A}$  on  $w$ .

In order to reason about Parikh images, we propose the following translation.

► **Lemma 11.** *Given an  $I/O$  transducer  $\mathcal{T}$ , we can construct two PRAs  $\mathcal{A}, \mathcal{B}$  over the alphabet  $2^I$  and with reward function of dimension  $k = |I|$ , such that for every  $x \in (2^I)^+$  and for every  $\mathbf{a} \in \mathbb{N}^k$ , we have that  $\Pr(\mathcal{A}(w) = \mathbf{a}) = \Pr(\mathfrak{P}(\mathcal{T}(x)) = \mathbf{a})$ , and  $\Pr(\mathcal{B}(w) = \mathbf{a}) = \Pr(\mathfrak{P}(\mathcal{T}(\pi(x))) = \pi(\mathbf{a}))$ .*

**Proof.** The translation is similar to the one given in the proof of Theorem 5, where instead of adding  $2^O$  to the alphabet, we collate the Parikh image using the rewards.

Let  $\mathcal{T} = \langle I, O, S, s_0, \delta, \ell \rangle$ , we construct  $\mathcal{A} = \langle S, 2^I, \delta, s_0, S \rangle$  with the following reward function: for every  $s \in S$  and  $1 \leq j \leq k$ , we have  $R(s)_j = 1$  if  $o_j \in \ell(s)$  and  $R(s)_j = 0$  otherwise (that is,  $R(s)$  is the characteristic vector of  $\ell(s)$ ). Thus,  $\mathcal{A}$  is identical to  $\mathcal{T}$ , where we treat all states as accepting, and replace output labels with their characteristic vectors.

The construction of  $\mathcal{B}$  is similar, but accounts for the permutation  $\pi$ : we define  $\mathcal{B} = \langle S, 2^I, \mu, s_0, S \rangle$  with reward function  $R'$ , where  $\mu(s, \mathbf{i}) = \delta(s, \pi(\mathbf{i}))$  for every state  $s \in S$  and  $\mathbf{i} \in 2^I$ , and  $R'(s) = \pi(R(s))$  (where  $R$  is the reward function of  $\mathcal{A}$ ). It is easy to see that the construction of  $\mathcal{A}$  and  $\mathcal{B}$  satisfies the conditions of the lemma. ◀

In [10], the problems of distribution-equivalence and expected-equivalence are solved, with complexities NC and RNC, respectively, where NC is the class of problems solvable using circuits of polynomial size and polylogarithmic depth, and RNC is its randomized analogue. It is known that  $\text{NC} \subseteq \text{P}$  and  $\text{RNC} \subseteq \text{RP}$ .

The distribution-equivalence and expected-equivalence problems, applied to the automata  $\mathcal{A}$  and  $\mathcal{B}$  obtained as per Lemma 11, exactly correspond to  $\pi$ -distribution symmetry and  $\pi$ -expected symmetry of  $\mathcal{T}$ , respectively. We thus have the following.

<sup>4</sup> The rewards in [10] also allow  $-1$  rewards, and is set on the transitions of the PRA. Since it is trivial to push rewards from the states to the transitions, our model is simpler.

► **Theorem 12.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a permutation  $\pi$ , whether it is  $\pi$ -Parikh distribution symmetric (resp.  $\pi$ -Parikh expected symmetric), is in NC (resp. RNC).*

Both notions of Parikh symmetry can be easily shown respect composition, analogously to Lemma 2, in that if  $\mathcal{T}$  is both  $\pi$ - and  $\tau$ - Parikh distribution/expected symmetric, then it is also  $\pi \circ \tau$ -Parikh distribution/expected symmetric. Thus, we conclude this section with the following.

► **Theorem 13.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a finite set of generators  $X = \{\pi_1, \dots, \pi_m\}$ , whether it is  $\pi$ -Parikh distribution symmetric (resp.  $\pi$ -Parikh expected symmetric) for every  $\pi \in \langle X \rangle$ , is in NC (resp. RNC).*

## 5 Qualitative Symmetry

Section 4.1 rules out a decidable quantitative approximation for symmetry that takes into account the order of the input (at least in the sense of Theorem 8). In lieu of such an approximation, we turn to study a qualitative approximation, whereby we only require that permuting the input does not alter the support of the output distribution.

Let  $\mathcal{T}$  be an I/O transducer, and let  $\pi \in \mathcal{S}_k$ . We say that  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric if for every  $x \in (2^I)^+$  and  $y \in (2^O)^+$  we have that  $\Pr(\mathcal{T}(x) = y) > 0$  iff  $\Pr(\mathcal{T}(\pi(x)) = \pi(y)) > 0$ .

Observe that for every  $x$  and  $y$  as above,  $\Pr(\mathcal{T}(x) = y) > 0$  iff there exists a run of  $\mathcal{T}$  on  $x$  that is labelled  $y$ . Thus, in order to study qualitative symmetry, we can ignore the concrete probabilities in  $\mathcal{T}$ , and only keep information on whether they are positive or not. Therefore, we essentially consider a nondeterministic transducer.

Using a similar translation to that in Theorem 5, but to NFAs instead of PAs, we have the following.

► **Lemma 14.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a permutation  $\pi$ , whether  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric, is in PSPACE.*

**Proof.** Similarly to our approach in Theorem 5, we translate  $\mathcal{T}$  to two automata  $\mathcal{A}$  and  $\mathcal{B}$ , where  $\mathcal{A}$  mimics the operation of  $\mathcal{T}$ , and  $\mathcal{B}$  works similarly, but under the permutation  $\pi$ . Then, we check the equivalence of  $\mathcal{A}$  and  $\mathcal{B}$ . Instead of using PAs, however, we now use nondeterministic automata (NFAs). An NFA is  $\mathcal{N} = \langle Q, \Sigma, \delta, q_0, F \rangle$  where  $Q$  is a set of states,  $\Sigma$  is an alphabet,  $\delta : Q \times \Sigma \rightarrow 2^Q$  is a transition function,  $q_0$  is an initial state, and  $F$  are the accepting states. The semantics of NFAs are textbook standard.

Let  $\mathcal{T} = \langle I, O, S, s_0, \delta, \ell \rangle$ . We define  $\mathcal{A} = \langle S, 2^{I \cup O}, \eta, s_0, S \rangle$  and  $\mathcal{B} = \langle S, 2^{I \cup O}, \zeta, s_0, S \rangle$ , where the transition functions are defined as follows. Let  $q \in S$  and  $\sigma = \mathbf{i} \cup \mathbf{o}$  with  $\mathbf{i} \in 2^I$  and  $\mathbf{o} \in 2^O$ , then  $\eta(q, \sigma) = \{p \in S : \delta(q, \mathbf{i})(p) > 0 \text{ and } \ell(p) = \mathbf{o}\}$  and  $\zeta(q, \sigma) = \{p \in S : \delta(q, \pi(\mathbf{i}))(p) > 0 \text{ and } \ell(p) = \pi(\mathbf{o})\}$ .

By construction, for every  $x \in (2^I)^+$  and  $y \in (2^O)^+$  we have that  $\Pr(\mathcal{T}(x) = y) > 0$  iff  $\mathcal{A}$  accepts  $x \otimes y$ , and  $\Pr(\mathcal{T}(\pi(x)) = \pi(y))$  iff  $\mathcal{B}$  accepts  $x \otimes y$ . Thus, we have that  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric iff  $L(\mathcal{A}) = L(\mathcal{B})$ . Since equivalence of NFAs can be checked in PSPACE, we are done. ◀

We proceed to show a matching lower bound.

► **Lemma 15.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a permutation  $\pi$ , whether  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric, is PSPACE-hard.*

**Proof.** We show the problem is PSPACE-hard via a reduction from the universality problem for NFAs over alphabet  $\Sigma = \{0, 1\}$  whose states are all accepting. That is, the problem of deciding, given an NFA  $\mathcal{A} = \langle Q, \{0, 1\}, \delta, q_0, Q \rangle$  (where all states are accepting), whether  $L(\mathcal{A}) = \Sigma^*$ . This problem was shown to be PSPACE-hard in [9].

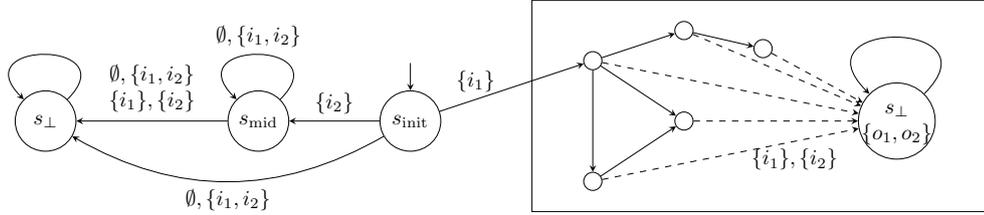
The reduction has a similar flavour as that of Theorem 8, in that we use the permutation to switch between components of the transducer. The components themselves, however, are somewhat different.

Let  $\mathcal{A} = \langle Q, \{0, 1\}, \delta, q_0, Q \rangle$  be an NFA over  $\{0, 1\}$  with all states accepting. We construct a transducer  $\mathcal{T} = \langle I, O, S, s_0, \eta, \ell \rangle$  over  $I = \{i_1, i_2\}$  and  $O = \{o_1, o_2\}$  as follows. The states are  $S = Q \cup \{s_{\text{init}}, s_{\text{mid}}, s_{\perp}\}$ , with the labelling  $\ell(q) = \emptyset$  for every  $q \in Q$ ,  $\ell(s_{\text{init}}) = \ell(s_{\text{mid}}) = \emptyset$ , and  $\ell(s_{\perp}) = \{o_1, o_2\}$ . For simplicity, we treat the transition function as nondeterministic  $\eta : S \times 2^{I \cup O} \rightarrow 2^S$ . Technically, this can be thought of as specifying the support of the transition function, with arbitrarily chosen probabilities (e.g., uniform). Note, however, that we do not allow  $\emptyset$  in the image of  $\delta$ , since we must be able to specify probabilities for the transitions. Now, for every  $q \in Q$  and  $\mathbf{i} \in 2^I$ , and we define

$$\eta(q, \mathbf{i}) = \begin{cases} \delta(q, 0) \cup \{s_{\perp}\} & \text{if } \mathbf{i} = \emptyset \\ \delta(q, 1) \cup \{s_{\perp}\} & \text{if } \mathbf{i} = \{i_1, i_2\} \\ \{s_{\perp}\} & \text{otherwise} \end{cases}$$

That is, within the  $Q$  component, we identify  $\Sigma = \{0, 1\}$  with  $\{\emptyset, \{i_1, i_2\}\}$ , and whenever there are no corresponding transitions in  $\mathcal{A}$ , or an “invalid” letter is seen, a transition is taken to  $s_{\perp}$ . Note that we add transitions to  $s_{\perp}$  even when there are transition in  $\mathcal{A}$ , which will play a role later on. The remaining transitions are as follows (see Figure 4).

$$\begin{aligned} \eta(s_{\text{init}}, \{i_1\}) &= \{q_0\}, & \eta(s_{\text{init}}, \{i_2\}) &= \{s_{\text{mid}}\}, \\ \eta(s_{\text{init}}, \emptyset) &= \eta(s_{\text{init}}, \{i_1, i_2\}) = \{s_{\perp}\}, & \eta(s_{\text{mid}}, \emptyset) &= \eta(s_{\text{mid}}, \{i_1, i_2\}) = \{s_{\text{mid}}, s_{\perp}\}, \\ \eta(s_{\text{mid}}, \{i_1\}) &= \eta(s_{\text{mid}}, \{i_2\}) = \{s_{\perp}\}, & \text{and } \eta(s_{\perp}, \sigma) &= \{s_{\perp}\}. \end{aligned}$$



■ **Figure 4** The transducer constructed from an NFA.

Let  $\pi = (1\ 2)$ . We claim that  $L(\mathcal{A}) = \Sigma^*$  iff  $\mathcal{T}$  is  $(1\ 2)$ -qualitative-symmetric.

For the first direction, we prove the contrapositive. Assume  $L(\mathcal{A}) \neq \Sigma^*$ , and let  $w \in \Sigma^* \setminus L(\mathcal{A})$ . Keeping our identification of  $\Sigma = \{0, 1\}$  with  $\{\emptyset, \{i_1, i_2\}\}$ , consider the word  $x = \{i_1\} \cdot w$ . Since there are no runs of  $\mathcal{A}$  on  $w$ , it follows that within the  $Q$  component, after reading  $w$ , the only reachable state is  $s_{\perp}$ . Thus, if  $z \in (2^O)^+$  is such that  $\Pr(\mathcal{T}(x) = z) > 0$ , then  $z$  is of the form  $\emptyset^+ \cdot \{o_1, o_2\}^+$ . In particular, let  $y = \emptyset^{|w|+1}$ , then  $\Pr(\mathcal{T}(x) = y) = 0$ . However, a possible run of  $\mathcal{T}$  on  $\pi(x)$  is  $s_{\text{init}}, s_{\text{mid}}^{|w|}$ , which induces the labels  $y = \pi(y)$ . Thus,  $\Pr(\mathcal{T}(\pi(x)) = \pi(y)) > 0$ , so  $\mathcal{T}$  is not  $\pi$ -qualitative-symmetric.

Conversely, assume that  $L(\mathcal{A}) = \Sigma^*$ , and consider  $x \in (2^I)^+$  and  $y \in (2^O)^+$ . We claim that  $\Pr(\mathcal{T}(x) = y) > 0$  iff  $\Pr(\mathcal{T}(\pi(x)) = \pi(y)) > 0$ . Observe that similarly to Theorem 8, all the labels on  $\mathcal{T}$  are invariant under  $\pi$ , so the above can be stated as

$$\Pr(\mathcal{T}(x) = y) > 0 \text{ iff } \Pr(\mathcal{T}(\pi(x)) = y) > 0. \quad (2)$$

Now, if  $x$  starts with either  $\emptyset$  or  $\{i_1, i_2\}$ , then there is a single run on  $x$  and on  $\pi(x)$ , namely  $s_{\text{init}}, s_{\perp}$ , so both  $x$  and  $\pi(x)$  induce the same distribution on output sequences. Thus, Equation (2) holds.

Next, similarly to Theorem 8, we can again assume without loss of generality that  $x$  starts with  $\{i_1\}$ , otherwise we use  $\pi(x)$ . Thus,  $x$  is either of the form  $\{i_1\} \cdot w$  or of the form  $\{i_1\} \cdot w \cdot \{\{i_1\}, \{i_2\}\} \cdot (2^I)^*$  with  $w \in \{\emptyset, \{i_1, i_2\}\}^*$ .

In the former case, recall that  $\eta$  follows the transition function of  $\mathcal{A}$ , as well as allowing at each point to reach  $s_{\perp}$ . Thus,  $\mathcal{T}(x)$  assigns positive probability to every word of the form  $\emptyset^+ \{o_1, o_2\}^*$  (of length  $|w| + 1$ ). Observe that  $\pi(w) = w$ , and hence  $\pi(x) = \{i_2\}w$ , which induces a distribution with the same support, and again Equation (2) holds.

In the latter case,  $x$  is of the form  $\{i_1\} \cdot w \cdot \{\{i_1\}, \{i_2\}\} \cdot (2^I)^*$ , where upon reading either  $\{i_1\}$  or  $\{i_2\}$ , the runs in the  $Q$  component all collapse to  $s_{\perp}$ . Thus, the support of  $\mathcal{T}(x)$  comprises words of the form  $\emptyset^+ \{o_1, o_2\}^*$  where the  $\emptyset^+$  prefix is at most of length  $|w| + 1$ . Since  $\pi(\{i_1\}) = \{i_2\}$  and  $\pi(\{i_2\}) = \{i_1\}$ , then by the definition of  $\eta$ , the distribution  $\mathcal{T}(\pi(x))$  has the same support (as runs that remain in  $s_{\text{mid}}$  collapse to  $s_{\perp}$  at the same stage). We thus conclude the claim. Finally, it is easy to see that the reduction is polynomial. ◀

Combining Lemmas 14 and 15, we have the following.

► **Theorem 16.** *The problem of deciding, given an I/O transducer  $\mathcal{T}$  and a permutation  $\pi$ , whether  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric, is PSPACE-complete.*

As in Section 4, since we use the permutation group  $\mathcal{S}_2$  for our hardness result, we have the following.

► **Corollary 17.** *The problem of deciding whether a given I/O transducer  $\mathcal{T}$  is  $\pi$ -qualitative-symmetric for every  $\pi \in \mathcal{S}_k$  is PSPACE-complete.*

## 6 Extensions and Research Directions

### Extensions

The setting considered thus far restricts to corresponding input and output sets of the form  $I = \{i_1, \dots, i_k\}$  and  $O = \{o_1, \dots, o_k\}$ . Typically, however, systems also include signals that are not process-specific, such as whether the system is ready, whether there is an error, etc. We can easily incorporate these into the setting. Indeed, adding input signals that are ignored by permutations can be inserted *mutatis-mutandis* to all the automata constructions we use. In addition, the lower bounds trivially carry over.

In addition, some systems have multiple sets of inputs and/or output signals that belong to processes, such as read grants and write grants, both of which are process-specific outputs. Again, our framework can easily be fit with this extension, by permuting each collection of process-specific inputs or outputs separately.

### Research Directions

Process symmetry often arises in model checking, and exploiting it correctly can significantly reduce the size of specifications (and hence the time spent in model checking), as well as give insight into the behaviour of the system. In this work, we introduce several variants of process symmetry, and study their algorithmic aspects. Specifically, we show that exact symmetry can be decided in polynomial time, whereas the approximate version via the  $L_{\infty}$  metric becomes undecidable. A coarser, qualitative approximation, can be decided in PSPACE. In addition, a different type of symmetry, which looks only at the Parikh image of the output, can be decided efficiently.

The notions of symmetry studied in this work restrict to either letter-by-letter symmetry, or Parikh symmetry. However, many other directions can exploit the structure of words as temporal objects to define other symmetry measures. These include *eventual symmetry*, where we require symmetry to take place only after a finite prefix, *sliding-window symmetry*, where we look at Parikh images within a sliding window, while requiring window-by-window symmetry, as well as notions of symmetry that are only relevant for infinite words, such as the limit-average Parikh image.

---

## References

- 1 Thomas Ball and Orna Kupferman. Vacuity in testing. In *International Conference on Tests and Proofs*, pages 4–17. Springer, 2008.
- 2 Peter J Cameron et al. *Permutation groups*, volume 45. Cambridge University Press, 1999.
- 3 Edmund M. Clarke, Reinhard Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1-2):77–104, 1996.
- 4 Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.
- 5 A Donaldson and Alice Miller. Symmetry reduction for probabilistic systems. In *Proc. 12th workshop on Automated Reasoning*, pages 17–18, 2005.
- 6 E Allen Emerson and A Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1-2):105–131, 1996.
- 7 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *International Colloquium on Automata, Languages, and Programming*, pages 527–538. Springer, 2010.
- 8 C Norris Ip and David L Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996.
- 9 Jui-Yi Kao, Narad Rampersad, and Jeffrey Shallit. On nfas where all states are final, initial, or both. *Theoretical Computer Science*, 410(47-49):5010–5021, 2009.
- 10 Stefan Kiefer and Björn Wachter. Stability and complexity of minimising probabilistic automata. In *International Colloquium on Automata, Languages, and Programming*, pages 268–279. Springer, 2014.
- 11 Marta Kwiatkowska, Gethin Norman, and David Parker. Symmetry reduction for probabilistic model checking. In *International Conference on Computer Aided Verification*, pages 234–248. Springer, 2006.
- 12 Anthony W Lin, Truong Khanh Nguyen, Philipp Rümmer, and Jun Sun. Regular symmetry patterns. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 455–475. Springer, 2016.
- 13 Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.
- 14 Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 2014.
- 15 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961.
- 16 A Prasad Sistla, Viktor Gyuris, and E Allen Emerson. Smc: a symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(2):133–166, 2000.
- 17 Corinna Spermann and Michael Leuschel. Prob gets nauty: Effective symmetry reduction for b and z models. In *2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 15–22. IEEE, 2008.
- 18 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.
- 19 Thomas Wahl and Alastair Donaldson. Replication and abstraction: Symmetry in automated formal verification. *Symmetry*, 2(2):799–847, 2010.

# Reachability in Dynamical Systems with Rounding

**Christel Baier** 

Technische Universität Dresden, Germany

**Florian Funke** 

Technische Universität Dresden, Germany

**Simon Jantsch** 

Technische Universität Dresden, Germany

**Toghrul Karimov** 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

**Engel Lefaucheu** 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

**Joël Ouaknine** 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany  
Department of Computer Science, Oxford University, UK

**Amaury Pouly** 

Université de Paris, CNRS, IRIF, F-75006, Paris, France

**David Purser** 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

**Markus A. Whiteland** 

Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

---

## Abstract

We consider reachability in dynamical systems with discrete linear updates, but with fixed digital precision, i.e., such that values of the system are rounded at each step. Given a matrix  $M \in \mathbb{Q}^{d \times d}$ , an initial vector  $x \in \mathbb{Q}^d$ , a granularity  $g \in \mathbb{Q}_+$  and a rounding operation  $[\cdot]$  projecting a vector of  $\mathbb{Q}^d$  onto another vector whose every entry is a multiple of  $g$ , we are interested in the behaviour of the orbit  $\mathcal{O} = \langle [x], [M[x]], [M[M[x]]], \dots \rangle$ , i.e., the trajectory of a linear dynamical system in which the state is rounded after each step. For arbitrary rounding functions with bounded effect, we show that the complexity of deciding point-to-point reachability – whether a given target  $y \in \mathbb{Q}^d$  belongs to  $\mathcal{O}$  – is **PSPACE**-complete for hyperbolic systems (when no eigenvalue of  $M$  has modulus one). We also establish decidability without any restrictions on eigenvalues for several natural classes of rounding functions.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** dynamical systems, rounding, reachability

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.36

**Related Version** A full version of the paper is available at <http://arxiv.org/abs/2009.13353>.

**Funding** This work was funded by DFG grant 389792660 as part of TRR 248 (see <https://perspicuous-computing.science>), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), DFG-projects BA-1679/11-1 and BA-1679/12-1, and the Research Training Group QuantLA (GRK 1763).

*Joël Ouaknine*: Supported by ERC grant AVS-ISS (648701).

*Amaury Pouly*: Supported by CODYS project ANR-18-CE40-0007.



© Christel Baier, Florian Funke, Simon Jantsch, Toghrul Karimov, Engel Lefaucheu, Joël Ouaknine, Amaury Pouly, David Purser, and Markus A. Whiteland;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 36; pp. 36:1–36:17



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

## 1 Introduction

A *discrete-time linear dynamical system* in ambient space  $\mathbb{Q}^d$  is specified via a linear transformation together with a starting point. The state of the system is then updated at each step by applying the linear transformation, giving rise to an *orbit* (or infinite trajectory) in  $\mathbb{Q}^d$ .

One of the most well-known questions for such systems is the *Skolem Problem*, which asks whether the orbit ever hits a given  $(d - 1)$ -dimensional hyperplane.<sup>1</sup> This problem has long eluded decidability, although instances of dimension  $d \leq 4$  are known to be solvable (see, e.g., the survey [30]). Another natural problem is *point-to-point reachability*<sup>2</sup>, known to be decidable in polynomial time [22]. In both cases, however, one assumes *arbitrary* precision, which arguably is unrealistic for simulations carried out on digital computers. In this paper, we therefore turn our attention to instances of these problems in which the numerical state of the system is rounded to finite precision at each time step. This leads us to the following definition:

► **Problem (Rounded Point-to-Point Reachability (Rounded P2P)).** *Given a matrix  $M \in \mathbb{Q}^{d \times d}$ , an initial vector  $x \in \mathbb{Q}^d$ , a target vector  $y \in \mathbb{Q}^d$ , a granularity  $g \in \mathbb{Q}_+$ , and a rounding operation  $[\cdot]$  projecting a vector of  $\mathbb{Q}^d$  onto another vector whose every entry is a multiple of  $g$ , let the orbit  $\mathcal{O}$  of this system be the infinite sequence  $\langle [x], [M[x]], [M[M[x]]], \dots \rangle$ , i.e.,  $x^{(0)} = [x]$  and  $x^{(i+1)} = [Mx^{(i)}]$ . The **Rounded Point-to-Point Reachability (Rounded P2P) Problem** asks whether  $[y] \in \mathcal{O}$ .*

**Main contributions.** We make the following contributions, summarised in Figure 1:

1. We introduce a family of natural problems, Rounded P2P (parameterised by the rounding function), which to the best of our knowledge has not previously been studied.
2. We show that for hyperbolic systems (i.e., those whose associated linear transformation has no eigenvalue of modulus 1) the Rounded P2P Problem is solvable – and is in fact **PSPACE**-complete – for any “reasonable” (i.e., bounded-effect) rounding function. It is interesting to note, in contrast, that exact P2P reachability is known to be solvable in polynomial time. Our approach to solving the Rounded P2P Problem relies on the observation that, outside a ball of exponential size, the change in magnitude of the system state at each step dwarfs any effect due to rounding. It thus suffices to exhaustively examine the effect of the dynamics inside an exponentially bounded state space.
3. In the general case (without any restriction on the magnitude of eigenvalues), the effect of rounding may forever remain non-negligible, requiring a careful analysis. We have not been able to solve the problem in full generality, but we do provide a complete solution for certain natural classes of rounding functions. More precisely, assume that the linear transformation has been converted to Jordan normal form (now requiring us to work with complex algebraic numbers). We can then solve the Rounded P2P Problem under two natural classes of rounding functions:
  - (a) *Polar rounding functions:* given a complex number of the form  $Ae^{i\theta}$ , such functions round  $A$  and  $\theta$  independently. In such instances we can handle in **EXSPACE** all reasonable rounding functions on  $A$ , and what we view as the only natural rounding function on  $\theta$ .

<sup>1</sup> The Skolem Problem is usually formulated in terms of linear recurrence sequences, but is equivalent to the description given here.

<sup>2</sup> Historically this problem has been known as the *orbit problem*, however there are now multiple “orbit problems” (polytope reachability, hyperplane reachability, (semi-)algebraic set reachability,... etc.) and so we specify point-to-point reachability.

Rounding type	Hyperbolic Systems	No restrictions on eigenvalues	
		Jordan normal form (Note: no hardness)	General
Polar $Ae^{i\theta}$	<b>PSPACE</b> -complete, Section 3	<b>EXPSpace</b> , Section 4.1	Open but <b>PSPACE</b> -hard
Argand truncation or expansion		<b>EXPSpace</b> , Section 4.2	
Argand minimal error		Open (difficulties highlighted in Section 5)	
Arbitrary bounded-effect		Open (Open Problem 21)	

■ **Figure 1** Decidability and complexity table for the Rounded P2P Problem.

- (b) *Argand rounding*: given a complex number of the form  $a + bi$ , the *Argand truncation* will round  $a$  and  $b$  independently downwards (in magnitude), ensuring that the modulus never increases. Similarly, the *Argand expansion* (which rounds  $a$  and  $b$  independently upwards) guarantees that the modulus can only increase. Under such rounding functions, we show decidability in **EXPSpace**.
4. We highlight some limitations of our methods, identifying a simple but technically challenging open problem, which points to some of the key difficulties in solving the Rounded P2P Problem in full generality. More precisely, we consider minimal error rounding for a simple rotation in two-dimensional space, for which Rounded P2P is presently open.

► **Remark.** It is worth noting that the rounded versions of the Skolem Problem (does the rounded orbit ever hit a  $(d - 1)$ -dimensional hyperplane?) and the Positivity Problem (does the rounded orbit ever hit a  $d$ -dimensional half-space?) remain at least as hard as their exact integer counterparts, since over the integers rounding has no effect; the decidability of these problems therefore remains open. However, the rounded versions of reaching a bounded polytope or a bounded semialgebraic set (problems not known to be decidable in the exact setting [14, 4]) reduce to a finite number of Rounded P2P reachability queries (since a bounded set can contain only finitely many rounded points). These observations together motivate our focus, in the present paper, on the Rounded P2P Problem.

It is interesting to consider rounded reachability problems in the stochastic setting, i.e., Markov chains. One observes that the state space  $[\mathcal{S}] = \{[x] \in [0, 1]^d \mid x \text{ sub-stochastic}\}$  is finite, which entails decidability of virtually any reachability problem, including Skolem and Positivity. This is somewhat arresting, since without rounding reachability problems are known to be exactly as hard for stochastic systems as for general systems [3]. In any event, one should note that ensuring that for all  $x \in [\mathcal{S}]$ ,  $[Mx] \in [\mathcal{S}]$  requires some care, as arbitrary rounding does not necessarily preserve (sub-)stochasticity.

## Related work

With the emerging use of numerical computations during the 80s, doubts were raised concerning the transferability of results about dynamical systems obtained by simulation in finite-state machines. In this direction, the sensitivity that a rounding function may have on the long-term behaviour of a dynamical system is studied in [5]. How rounded orbits can be simulated by actual orbits of the dynamical system is investigated in [20, 29].

The series of papers [6, 7, 8, 9] examines which statistical properties of a discrete dynamical system are preserved under the introduction of a rounding function, a good summary of which can be found in Blank's book [10, Chapter 5]. As the rounding is refined, some properties

of the discretized orbits follow probabilistic laws asymptotically, as shown in [16, 17]. The paper [18] studies how volatile statistical notions are in the presence of finite precision (such as the mean distance of two orbits of discrete dynamical systems).

Another line of research focuses on discretized rotations in  $\mathbb{Z}^2$  and higher-dimensional lattices [24, 1]. A connection from roundoff problems in the 2-dimensional case to expanding maps on the  $p$ -adic integers is described in [11, 36]. Building on this, [35] conjectures periodicity of all orbits of these discretized rotations in  $\mathbb{Z}^2$ . It is shown in [2] that there are infinitely many periodic orbits, and [31] attempts to concisely describe points leading to periodic orbits.

In the context of model checking, continuous dynamical systems have been translated into discrete models, mainly timed automata that approximate the behaviour of the original system [26, 13, 32]. On a more general level, one can observe a growing interest in the systematic study of roundoff errors inherent in finite precision computations [19, 33, 21, 25, 27, 15].

## 2 Rounding functions

Let  $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{A}$  be the naturals, integers, rationals, reals, and algebraic numbers respectively.

### Rounding real numbers

Let  $g \in \mathbb{R}_+$  be a granularity. We define our rounding functions taking values to integers, i.e.,  $g = 1$ . For  $g \neq 1$  we consider  $\lfloor x \rfloor = g \cdot \lfloor x/g \rfloor$ . Given a set  $S$ , we let  $\lfloor S \rfloor = \{\lfloor x \rfloor \mid x \in S\}$ .

The floor function  $\lfloor x \rfloor$  and ceiling functions  $\lceil x \rceil$  are well-known rounding functions in mathematics and computer science. We recall two further rounding functions:

- *Minimal error rounding* rounds to the *nearest* value:  $\lfloor x \rfloor = \arg \min_{y \in \mathbb{Z}} |x - y|$ . If  $|x - y| = 0.5$  an arbitrary but deterministic choice must be made (e.g. to round up).
- *Truncation* (“towards zero rounding”, to cut off the remaining bits): if  $x > 0$  then  $\lfloor x \rfloor$  else  $\lceil x \rceil$ , or *expansion*: if  $x > 0$  then  $\lceil x \rceil$  else  $\lfloor x \rfloor$ .

Whenever possible, we prefer to analyse the problems without choosing a specific rounding function, relying only upon the property of *bounded effect*:

► **Definition 1.** A real rounding function  $\lfloor \cdot \rfloor: \mathbb{R} \rightarrow \mathbb{R}$  has bounded effect if there exists  $\Delta$  such that  $|x - \lfloor x \rfloor| \leq \Delta$  for all  $x$ .

### Rounding complex numbers

Complex numbers have both a real and imaginary part. Thus one can consider rounding each of the components separately, which we call *Argand rounding*. Consider  $x = a + bi$  with  $a, b \in \mathbb{R}$ , then let  $\lfloor x \rfloor = \lfloor a \rfloor + \lfloor b \rfloor i$ , where  $\lfloor \cdot \rfloor$  can be any real rounding function (leading to *Argand truncation*, *Argand expansion* and *Argand minimal error rounding* functions).

However, complex numbers can also be readily represented using polar coordinates as follows: a number is represented as  $x = Ae^{i\theta}$ , where  $A$  is the modulus and  $\theta$  is the angle between the 2-d coordinates  $(1, 0)$  and  $(a, b)$  (when represented as  $a + bi$ ). Then, a *polar rounding* function rounds  $A$  and  $\theta$  independently, i.e.  $\lfloor x \rfloor = \lfloor A \rfloor e^{i\lfloor \theta \rfloor}$ . The rounding of  $\lfloor A \rfloor$  can be any real rounding function. For the rounding of the angle we always assume minimal error rounding. That is, given granularity  $\theta_g = \frac{\pi}{R}$  for some  $R \in \mathbb{N}$ , then  $\lfloor \theta \rfloor$  is a multiple of  $\theta_g$  with minimal error and arbitrary but deterministic tie breaking.

We generalise non-specific bounded-effect rounding to the complex numbers.

► **Definition 2.** A complex rounding function  $\lfloor \cdot \rfloor: \mathbb{C} \rightarrow \mathbb{C}$  has bounded effect on the modulus if there exists  $\Delta$  such that  $||x| - |\lfloor x \rfloor|| \leq \Delta$  for all  $x$ .

Argand and polar roundings are both defined by applying bounded-effect real rounding functions to each component, and have bounded effect under Definition 2. However, note the distinction with Definition 1; polar rounding can exhibit arbitrary large effects (in the following sense: given any  $\Delta > 0$ , one can always find  $x \in \mathbb{C}$  such that  $|x - [x]| > \Delta$ ), but nevertheless has only bounded effect *on the modulus*.

► **Definition 3** ( $[K]$ -Ball). *Given a complex rounding function  $[\cdot]$  and an integer  $K$  let a  $[K]$ -ball be the set of admissible points of modulus at most  $K$ , i.e.,  $\{[x] \mid x \in \mathbb{C}, |[x]| \leq K\}$ .*

### Rounding vectors

In general, a rounding function on  $\mathbb{K}$  induces a rounding function on vectors  $\mathbb{K}^d$ , where  $[(x_1, \dots, x_d)] = ([x_1], \dots, [x_d])$ , although not all rounding functions on vectors need take this form. We generalise non-specific bounded-effect rounding to vectors.

► **Definition 4.** *A rounding function  $[\cdot]: \mathbb{K}^d \rightarrow \mathbb{K}^d$  has bounded effect on the modulus if there exists  $\Delta$  such that  $||x|_k - |[x]|_k| \leq \Delta$  for all  $x$  and every  $k \in \{1, 2, \dots, d\}$ .*

Finally, we assume that all of our rounding functions can be computed in polynomial time and are fixed (rather than inputs) in our problems, and thus  $\Delta$  is also a fixed parameter.

## 3 Hyperbolic systems

In this section we establish our first main result for hyperbolic systems, which we first define:

► **Definition 5** (Hyperbolic System [23, Section 1.2]). *A linear map represented by the matrix  $M \in \mathbb{R}^{d \times d}$  is hyperbolic if all of its eigenvalues have modulus different from one.*

► **Theorem 6.** *The Rounded P2P Problem is **PSPACE**-complete for hyperbolic linear maps represented by rational matrices and real rounding functions with bounded effect.*

We first demonstrate that the problem is in **PSPACE** for matrices in Jordan normal form, to which we will reduce the general case in a second step. As the passage to Jordan normal form inevitably introduces complex numbers, **PSPACE** membership will be shown for Jordan normal form matrices over the algebraic numbers and, accordingly, complex rounding functions with bounded effect on the modulus. To complete the picture we show hardness for hyperbolic systems (in fact, the hardness result applies even for non-hyperbolic systems, that is for matrices whose eigenvalues may include 1).

### 3.1 Membership in PSPACE

We now prove the membership part of Theorem 6 under the additional assumption that the matrices are in Jordan normal form.

► **Lemma 7.** *The Rounded P2P Problem decidable in **PSPACE** for any complex rounding function with bounded effect on the modulus  $\Delta$  and hyperbolic matrices  $M \in \mathbb{A}^{d \times d}$  in Jordan normal form.*

## 36:6 Reachability in Dynamical Systems with Rounding

**Proof.** We consider a single Jordan block of dimension  $d$  with eigenvalue  $\lambda$ . If the matrix  $M$  has multiple Jordan blocks, the algorithm can be run in lock step<sup>3</sup> for each block. Hence, without loss of generality we let

$$M = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & 1 & \\ & & \ddots & \ddots \\ & & & \lambda \end{bmatrix}.$$

The idea will be to show that for  $|\lambda| > 1$ , for values large enough growth will outstrip the rounding, and the orbit will grow beyond the target, never to return. If  $|\lambda| < 1$  and the orbit gets large enough, it will begin to contract again, so we choose a ball large enough to contain the whole orbit. We do not consider the case  $|\lambda| = 1$  here.

Formally, in each dimension  $k \in \{1, \dots, d\}$  we compute a radius  $C_k$ , defining a  $[C_k]$ -ball of radius  $C_k$  about 0, containing  $x_k$  and  $y_k$  such that for all  $z$  in the orbit  $\mathcal{O}$  if  $z_k \notin [C_k]$ -ball then  $[Mz]_k \notin [C_k]$ -ball. That is, if the orbit has left the ball, it will never come back. The algorithm proceeds by simulating the orbit from  $x$  until one of the following occurs.

- $y$  is found, in which case RETURN YES, or
- a point repeats, in which case RETURN NO, or
- a point  $x^{(i)}$  is found such that  $|(x^{(i)})_k| \geq C_k$  for some  $k$ , in which case RETURN NO.

Since  $B = \{x \in \mathbb{R}^d \mid \text{for all } k \ |x_k| \leq C_k\}$  is finite, one of the three must occur. Remembering all previous points would require too much space. Therefore we record a counter of the number of steps taken and once this exceeds the maximum number of points then we know some point must have been repeated (possibly many times by this point). Let  $C = \max_i C_k$ , then the bounding hyper-cube of  $B$  has  $(2C/g)^d$  points, hence  $B$  has fewer points. We show this number has at most exponential size in the description length of the input, and hence can be represented in **PSPACE**.

► **Case 1 (suppose  $|\lambda| > 1$ ).** For the  $d$ th component we have  $(x^{(i+1)})_d = [\lambda(x^{(i)})_d]$ . There is a bounded effect of the rounding  $\Delta$ , ensuring  $|(x^{(i+1)})_d| \geq |\lambda| |(x^{(i)})_d| - \Delta$ . So when  $|\lambda| |(x^{(i)})_d| - \Delta > |(x^{(i)})_d|$ , this component must grow. Let  $\ell = \max\{1, \Delta, |y_1|, \dots, |y_d|\}$ . We define the radius  $C_d := \frac{\Delta}{|\lambda|-1} + \ell$ , which satisfies the desired property described above.

Now suppose that the radius  $C_k$  is defined so that  $C_k \leq \ell \sum_{j=0}^{d-k+1} (\frac{2}{|\lambda|-1})^j$  (holds for  $k = d$ ) and assume that  $|(x^{(i)})_j| \leq C_j$  for each  $j \in \{k, \dots, d\}$ . For the  $k-1$ th dimension the update is of the form  $(x^{(i+1)})_{k-1} = [\lambda(x^{(i)})_{k-1} + 1(x^{(i)})_k]$ . Since  $|(x^{(i)})_k| \leq C_k$ , we have  $|(x^{(i+1)})_{k-1}| \geq |\lambda| |(x^{(i)})_{k-1}| - \Delta - C_k$ , and there is growth when  $|\lambda| |(x^{(i)})_{k-1}| - \Delta - C_k > |(x^{(i)})_{k-1}|$ , i.e., when  $|(x^{(i)})_{k-1}| > \frac{\Delta + C_k}{|\lambda|-1}$ . So, we may define  $C_{k-1} := \frac{\Delta + C_k}{|\lambda|-1} + \ell$ , which satisfies the property described above, and moreover,  $C_{k-1} \leq \frac{2C_k}{|\lambda|-1} + \ell \leq \ell \sum_{j=0}^{d-(k-1)+1} (\frac{2}{|\lambda|-1})^j$  due to our choice of  $\ell$ . Repeat for all remaining components  $k-2, \dots, 1$ .

Now  $C_k \leq \ell \sum_{j=0}^d (\frac{2}{|\lambda|-1})^j \leq \ell(d+1)(1 + (\frac{2}{|\lambda|-1})^d)$  for each  $k$ , and the claim follows.

► **Case 2 (suppose  $|\lambda| < 1$ ).** We require the ball to have the property that if the orbit leaves, it will never come back. However for  $|\lambda| < 1$ , while initially there may be some growth (due to other components), once large enough  $|\lambda|$  will dominate and the modulus will decrease. Therefore, we want to ensure we choose the ball large enough that the orbit will never leave the ball in the first place. The following definitions of the radii  $C_j$  can easily be altered to furnish this requirement.

<sup>3</sup> By running processes in lock step, here and elsewhere, we mean running all of the processes simultaneously (interleaving instructions for each process) until either  $x^{(i)} = y$  or one of the processes concludes non-reachability.

Consider the last component  $d$ : we have  $|(x^{(i+1)})_d| \leq |\lambda| |(x^{(i)})_d| + \Delta$ . Set again  $\ell = \max\{1, \Delta, |y_1|, \dots, |y_d|\}$  and define  $C_d := \frac{\Delta}{1-|\lambda|} + \ell$ ; if  $|(x^{(i)})_d| \leq C_d$ , then  $|(x^{(i+1)})_d| \leq C_d$ .

Having fixed  $C_{k'}$  for  $k' \in \{k, \dots, d\}$ , consider component  $k-1$ : We have  $(x^{(i+1)})_{k-1} = [\lambda(x^{(i)})_{k-1} + (x^{(i)})_k]$ , and so  $|(x^{(i+1)})_{k-1}| \leq |\lambda| |(x^{(i)})_{k-1}| + |(x^{(i)})_k| + \Delta$ . Let us define  $C_{k-1} := \frac{C_k + \Delta}{1-|\lambda|} + \ell$ . Now if  $|(x^{(i)})_{k-1}| \leq C_{k-1}$  then  $|(x^{(i+1)})_{k-1}| \leq C_{k-1}$ . Repeat for each remaining component. It can be shown, similar to the previous case, that  $C_k \leq \ell(d+1)(1 + (\frac{2}{1-|\lambda|})^d)$  for each  $k$ , and this concludes the proof.  $\blacktriangleleft$

### Reducing the general form to Jordan normal form

In the previous section we assumed that the matrix is always in Jordan normal form, which is a significant restriction. In this section we will not assume Jordan normal form, which means we cannot make any assumption about the rounding, other than being of bounded effect, to prove Theorem 6. After a change of basis properties such as “rounding towards zero” may not be preserved.

**Proof (upper bound of Theorem 6).** Let  $\Delta$  be the fixed, bounded effect on the modulus of  $[\cdot]$ . Let us consider hyperbolic  $M = PJP^{-1} \in \mathbb{Q}^{d \times d}$ . We ask whether  $x^{(i+1)} = y$  for some  $i$ . Observe that  $x^{(i+1)} = [Mx^{(i)}] = Mx^{(i)} + e(Mx^{(i)})$  where  $e(x) := [x] - x \in [-\Delta, \Delta]^d$  for any  $x$  since  $[\cdot]$  has bounded effect. Now if we define  $z^{(i)} := P^{-1}x^{(i)}$  we have that

$$z^{(i+1)} = P^{-1}x^{(i+1)} = P^{-1}(Mx^{(i)} + e(Mx^{(i)})) = Jz^{(i)} + P^{-1}e(PJz^{(i)}) = (\lceil Jz^{(i)} \rceil)$$

where  $(\lceil z \rceil) := z + P^{-1}e(Pz)$  for any  $z$ . The question  $x^{(i)} \stackrel{?}{=} y$  for some  $i$  now becomes equivalent to  $z^{(i)} \stackrel{?}{=} P^{-1}y$ . But note that the system for  $z^{(i)}$  is in Jordan normal form and the rounding function  $(\lceil \cdot \rceil)$  has bounded effect on the modulus, with bound  $\Delta' \leq \max_{1 \leq k \leq d} \max_{e \in [-\Delta, \Delta]^d} (P^{-1}e)_k$ . Since  $\Delta$  is fixed and  $P^{-1}$  is computable in polynomial time [12], then  $\Delta'$  is of polynomial size. Hence, we have produced in polynomial time an instance of the Rounded P2P problem with a matrix in Jordan normal form. As the proof of Lemma 7 shows that this problem is solvable in **PSPACE** even if  $\Delta$  is given as input, we can conclude that the **PSPACE** upper bound holds also for the general case.  $\blacktriangleleft$

## 3.2 PSPACE-hardness

We will prove **PSPACE**-hardness (i.e., the lower bound of Theorem 6) by reduction from quantified boolean formula (QBF), which is **PSPACE**-complete [34]. We do this by first encoding a simple programming language into the rounded P2P Problem. Then, we show that reachability in this language can solve QBF. Whilst a direct reduction is possible, we provide exposition via the language for two reasons; first, we will show that the language is robust to choice of rounding function (Remark 9), and secondly the reduction results in an instance where all eigenvalues have modulus 1, but by a small perturbation, we observe that the problem remains hard when all of the eigenvalues do not have modulus 1 (Remark 10).

The language will consist of  $m$  instructions, operating over  $d$  variables. Each instruction is a boolean map  $f_i : [0, 1]^d \rightarrow [0, 1]^d$ , where each dimension  $i$  is updated using a logical formula of the  $d$  inputs. Each of the  $m$  instructions is conducted in turn and updating the  $d$  variables is *simultaneous* in each step. Thus, references to variable in a function are the evaluation in the previous step. Once the  $m$  instructions are complete, the system returns to the first instruction and repeats  $(x^{(i)} = (f_m \circ f_{m-1} \circ \dots \circ f_2 \circ f_1)(x^{(i-1)}))$ , see also Algorithm 1).

An instruction is encoded into the rounded dynamical system using a map  $f_i : \mathbb{N}^d \rightarrow \mathbb{N}^d$  for  $0 \leq i \leq m-1$ , where instructions are of the form  $(f_i(x))_j = \lfloor (p_j \cdot x) \rfloor$  where  $p_j$  in  $\mathbb{Q}^d$ . We demonstrate how to encode the required logical operations in a rounded dynamical

■ **Algorithm 1** System behaviour of the language.

---

**Input:**  $x \in [0, 1]^d$  initial vector,  $y \in [0, 1]^d$  target vector

**while**  $x \neq y$  **do**

$x \leftarrow f_1(x)$	
$x \leftarrow f_2(x)$	$\stackrel{\text{e.g.}}{=} \begin{cases} x_1 \leftarrow x_2 \vee (x_5 \wedge x_3) \\ x_2 \leftarrow \text{if } (x_1 \vee x_3) \text{ then } x_6 \text{ else } x_2 \\ x_3 \leftarrow \text{true} \\ \vdots \\ x_d \leftarrow x_4 \end{cases}$
$\vdots$	
$x \leftarrow f_m(x)$	

**end**

---

system: and ( $x_i \leftarrow x_j \wedge x_k = \lfloor \frac{1+x_j+x_k}{3} \rfloor$ ), or ( $x_i \leftarrow x_j \vee x_k = \lfloor \frac{1+x_j+x_k}{2} \rfloor$ ), negation ( $x_i \leftarrow \neg x_j = \lfloor 1 - x_j \rfloor$ ), resetting a variable to **false** ( $x_i \leftarrow \lfloor 0 \rfloor$ ), copying a variable without change ( $x_i \leftarrow \lfloor x_j \rfloor$ ) or moving/duplicating a variable ( $x_i \leftarrow \lfloor x_j \rfloor$ ). To enable this, we will assume there is always access to the constant 1 (or **true**) by an implicit dimension, fixed to 1.

In multiple steps any logical formula can be evaluated. This can be done with auxiliary variables to store partial computations, where the instructions will in fact be multi-step instructions making use of a finite collection of auxiliary variables which will not be referenced explicitly. Meanwhile any unused variables can be copied without change. In particular the syntax  $x_1 \leftarrow \text{if } (x_2) \text{ then } x_3 \text{ else } x_4$  can be encoded, by equivalence with the logical formula  $x_1 \leftarrow ((x_2 \implies x_3) \wedge (\neg x_2 \implies x_4))$ .

We ask, given some initial configuration  $x^{(0)}$ , and a target  $y$ : does there exist  $i$  such that  $x^{(i)} = y$ . If there was just one step function, the system dynamics would be a direct instance of the rounded orbit semantics. When there are  $m$  functions, we remark the sequence of functions can be encoded by taking  $m$  copies of each variable, and each function  $f_i$ , can transfer the function from one copy to the next, zeroing the previous set of variables. That is, let

$$M = \begin{bmatrix} 0 & & & & f_m \\ f_1 & 0 & & & \\ & f_2 & 0 & & \\ & & \ddots & & \\ & & & f_{m-1} & 0 \end{bmatrix}.$$

Then the initial configuration becomes  $(x^{(0)}, 0, \dots, 0)$ , and the target becomes  $(y, 0, \dots, 0)$ .

An abstraction of the language is depicted in Algorithm 1. It remains to show that QBF can be encoded in the language.

► **Lemma 8.** *Reachability in this language can solve QBF.*

**Proof.** Formally we write a program in our language to decide the truth of a formula of the form  $\forall x_1 \exists x_2 \forall x_3 \dots \exists x_n \psi(x_1, \dots, x_n)$ , where  $\psi$  is a quantifier free boolean formula. For convenience we assume it starts with  $\forall$ , ends with  $\exists$  and alternates. Formulae not in this form can be padded if necessary with variables which do not occur in the formula  $\psi$ .

The program will have the following variables:  $x_1, \dots, x_n, \hat{\psi}, s_1^0, \dots, s_n^0, s_1^1, \dots, s_n^1$  and  $c_1, \dots, c_n$ . The bits  $x_1, \dots, x_n$  represent the current allocation to the corresponding bit

variables of  $\psi$ , and  $\hat{\psi}$  will store the current evaluation of  $\psi(x_1, \dots, x_n)$ . To cycle through all allocations to  $x_1, \dots, x_n$ , the variables will be treated as a binary number and incremented by one many times, for this purpose the bits  $c_1, \dots, c_n$  represent the carry bits when incrementing  $x_1, \dots, x_n$ .

The intuition of  $s_i^z$  is the following: for fixed  $x_1, \dots, x_{i-1}$  it stores the evaluation of  $Qx_{i+1} Q'x_{i+2} \dots \exists x_n \psi(x_1, \dots, x_{i-1}, z, x_{i+1}, \dots, x_n)$  where  $Q, Q' \in \{\exists, \forall\}$  as required by the formula. Therefore the overall formula is true if and only if  $s_1^0 \wedge s_1^1$  is eventually true.

We define  $3 + n$  instructions, and each run through  $f_1 \rightarrow f_{3+n}$  will cover exactly one allocation to  $x_1, \dots, x_n$ , with the next run through covering the next allocation that one gets by incrementing the rightmost bit. Once  $x_{i+1}$  has been in both the 1 state and the 0 state for all values below, we have enough information to set  $s_{i+1}^{x_i}$ . This is set when the carry-bit  $c_{i+1}$  is one, which indicates that  $x_{i+1}$  has visited both 0 and 1 and is being returned back to 0 (thus setting  $x_{i+1} = \dots = x_n$  back to 0).

We let the initial configuration be  $(0, 0 \dots, 0)$ . Note that this is hiding the implicit dimension that is always 1. Each of the following step functions should be interpreted as copying any variable that is not explicitly set.

<b>Step 1.</b>	<b>Step 2.</b>	<b>Step 3.</b>
Evaluate $\psi$	Update either $s_n^0$ or $s_n^1$	Start incrementing $x_n$
$f_1(\cdot) = \left\{ \hat{\psi} \leftarrow \psi(x_1, \dots, x_n) \right.$	$f_2(\cdot) = \left\{ \begin{array}{l} s_n^0 \leftarrow \text{if } (x_n = 0) \text{ then } \hat{\psi} \\ \qquad \qquad \qquad \text{else } s_n^0 \\ s_n^1 \leftarrow \text{if } (x_n = 1) \text{ then } \hat{\psi} \\ \qquad \qquad \qquad \text{else } s_n^1 \end{array} \right.$	$f_3(\cdot) = \left\{ \begin{array}{l} x_n \leftarrow \neg x_n \\ c_n \leftarrow x_n \end{array} \right.$

**Step 3 + n - i, for i = n - 1 to 1.**

If there is a carry, update  $s_i^z$  and continue incrementing

<b>i even (<math>x_i</math> universally quantified):</b>	<b>i odd (<math>x_i</math> existentially quantified):</b>
$f_{3+n-i}(\cdot) = \left\{ \begin{array}{l} x_i \leftarrow \text{if } (c_{i+1}) \text{ then } \neg x_i \text{ else } x_i \\ c_i \leftarrow c_{i+1} \wedge x_i \\ c_{i+1} \leftarrow 0 \\ s_i^0 \leftarrow \text{if } (c_{i+1} \wedge \neg x_i) \text{ then } s_{i+1}^0 \wedge s_{i+1}^1 \\ \qquad \qquad \qquad \text{else } s_i^0 \\ s_i^1 \leftarrow \text{if } (c_{i+1} \wedge x_i) \text{ then } s_{i+1}^0 \wedge s_{i+1}^1 \\ \qquad \qquad \qquad \text{else } s_i^1 \end{array} \right.$	$f_{3+n-i}(\cdot) = \left\{ \begin{array}{l} x_i \leftarrow \text{if } (c_{i+1}) \text{ then } \neg x_i \text{ else } x_i \\ c_i \leftarrow c_{i+1} \wedge x_i \\ c_{i+1} \leftarrow 0 \\ s_i^0 \leftarrow \text{if } (c_{i+1} \wedge \neg x_i) \text{ then } s_{i+1}^0 \vee s_{i+1}^1 \\ \qquad \qquad \qquad \text{else } s_i^0 \\ s_i^1 \leftarrow \text{if } (c_{i+1} \wedge x_i) \text{ then } s_{i+1}^0 \vee s_{i+1}^1 \\ \qquad \qquad \qquad \text{else } s_i^1 \end{array} \right.$

**Step 3 + n.**

Set every variable to 1 if QBF satisfied. After this step, the program returns to  $f_1$ .

$$f_{3+n}(\cdot) = \left\{ v \leftarrow \text{if } (s_1^0 \wedge s_1^1) \text{ then } 1 \text{ else } v \quad (\text{for all variables } v) \right.$$

The  $(3+n)$ th step ensures that configuration  $(1, \dots, 1)$  will be reached if and only if the given QBF formula is satisfied. ◀

► **Remark 9 (Choice of rounding function).** The presentation here relies on specific choices of rounding function, but we observe that the language can easily exchange several different natural rounding functions, so the reduction is robust. The rounding is only useful in the **and** and **or** instructions. The floor function can be replaced by essentially any other rounding. For example  $x_j \vee x_k = \left\lceil \frac{x_j + x_k}{2} \right\rceil$  and  $x_j \wedge x_k = \left\lfloor \frac{-1 + x_j + x_k}{2} \right\rfloor$ . Similarly, when  $\lceil \cdot \rceil$  is minimal error rounding then  $x_j \vee x_k = \left\lceil \frac{1 + x_j + x_k}{3} \right\rceil$  and  $x_j \wedge x_k = \left\lfloor \frac{x_j + x_k}{3} \right\rfloor$  (the break point is not used). Thus, the problem will also be hard for any of these roundings.

► **Remark 10** (Perturbation: ensuring the eigenvalues are not modulus 1). Observe that under the perturbation that multiplies each operation by 1.1 (before taking floor) we obtain the same resulting operation. For example  $x_i \leftarrow x_j \vee x_k = \left\lfloor \frac{1+x_j+x_k}{2} \right\rfloor$  is equivalent to  $x_i \leftarrow x_j \vee x_k = \left\lfloor \left( \frac{1+x_j+x_k}{2} \right) * 1.1 \right\rfloor$ . Hence, if the resulting matrix  $M$  has eigenvalues 1, taking  $1.1M$  (or similar value to 1.1) will result in a matrix that does not with the same orbit; which shows that hardness is retained for matrices in which no eigenvalue has modulus 1.

► **Remark 11** (Dimension). The hardness result needs reachability instances of unbounded dimension. For a QBF formula with  $n$  variables and  $\ell$  logical operations, the resulting instance of rounded P2P has dimension  $(3n + 1 + \ell)(4n + 15 + \ell)$ .

## 4 Special cases on non-hyperbolic systems

In this section we consider certain cases when the eigenvalues can be of modulus one. In particular we work in the Jordan normal form and show that the problem can be solved for certain types of rounding. We fall short of arbitrary deterministic rounding, which would be required to show the problem in full generality through the Jordan normal form approach.

First, we show decidability for polar-rounding, along with an example with numbers requiring exponential space by the time the system becomes periodic – seeming to imply any “wait and see” approach would require **EXPSpace**. We also show decidability for certain types of Argand rounding, in particular truncation and expansion, but minimal-error rounding remains open (which we discuss further in Section 5).

### 4.1 Polar rounding with updates in Jordan normal form

We restrict ourselves to a Jordan block  $M$  of dimension  $d$ , with eigenvalue  $\lambda$  of modulus 1. Since the polar rounding function has bounded effect *on the modulus*, the remaining blocks, which need not be of modulus 1 can be solved (Lemma 7) by running this algorithm in lock step with the algorithm for those blocks. All together, this gives us:

► **Theorem 12.** *The Rounded P2P Problem is decidable in **EXPSpace** for the polar rounding function with  $\theta_g = \frac{\pi}{R}$ ,  $R \geq 2$  and matrices  $M \in \mathbb{A}^{d \times d}$  in Jordan normal form.*

To prove Theorem 12 we show that each dimension  $d, d-1, \dots, 1$  will eventually be periodic on a fixed modulus, or permanently diverge beyond  $y_k$  (the target value in dimension  $k$ ).

Let  $\langle a, b \rangle$  be the smallest angle between vectors  $a$  and  $b$  – this is a value in  $[0, \pi]$  and, in particular, it is always positive. It is used as a measure of alignment: the more  $a$  and  $b$  are aligned the smaller  $\langle a, b \rangle$  is. We will assume that the system will round up if  $[x] - x = 0.5$ . The remaining case can be adapted by suitably adjusting the relevant inequalities. We say that a dimension  $k \in \{1, \dots, d\}$  is *just rotating* after position  $N$ , if for all  $i \geq N$ :  $(x^{(i+1)})_k = \lfloor \lambda (x^{(i)})_k \rfloor$ . Note that dimension  $d$  is just rotating after 0, by definition. Our goal is to show that every dimension  $k$  will eventually be just rotating (for which we would require it to have modulus  $|y_k|$ ) or reach a point that lets us conclude it has permanently diverged past  $y_k$ . So we assume, henceforth, that dimension  $k$  is just rotating.

We let  $\phi(i) = \langle \lambda (x^{(i)})_{k-1}, (x^{(i)})_k \rangle$ . As  $(x^{(i+1)})_{k-1} = \lfloor \lambda (x^{(i)})_{k-1} + (x^{(i)})_k \rfloor$ , small values of  $\phi(i)$  (between 0 and  $\pi/2$ ) lead to an increase in modulus of  $(x^{(i+1)})_{k-1}$ , whereas large values (between  $\pi/2$  and  $\pi$ ) lead to a decrease when  $|(x^{(i)})_{k-1}|$  is sufficiently large relative to  $|(x^{(i)})_k|$ . Our analysis relies on the fact that  $\phi(i)$  can never increase:

► **Lemma 13.** *Suppose that dimension  $k$  is just rotating after step  $N$ . Then, for all  $i \geq N+1$ :  $\phi(i) \geq \phi(i+1)$ .*

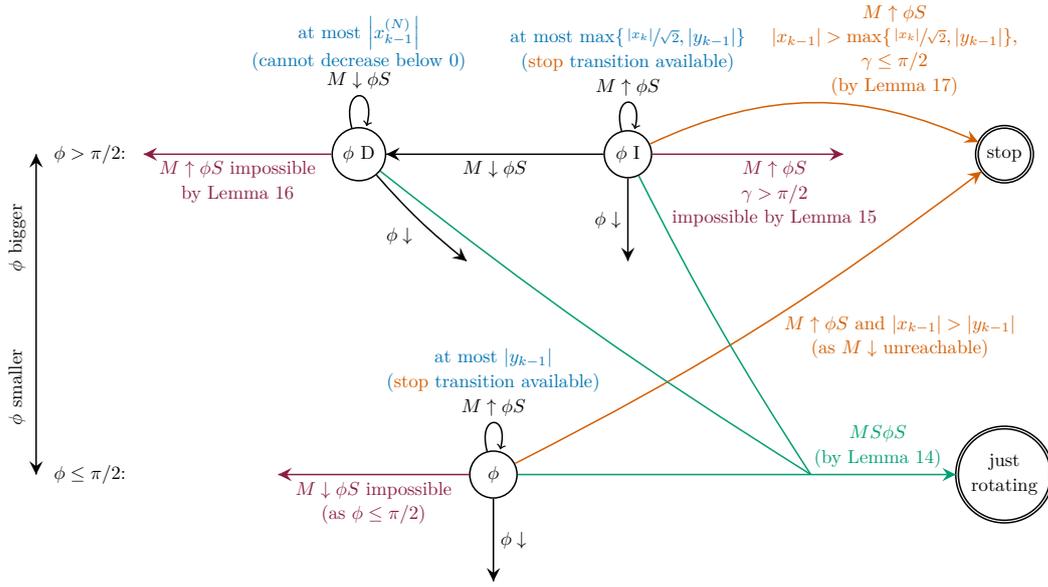


Figure 2 State diagram for  $\phi$  whilst considering dimension  $k-1$ , assuming  $k$  is just rotating.

If dimension  $k-1$  repeats its relative angle to  $k$  and its modulus in some step, we can conclude that  $k-1$  is just rotating:

► **Lemma 14.** *Suppose that dimension  $k$  is just rotating after step  $N$ , that  $\phi(N) = \phi(N+1)$  and  $|(x^{(N)})_{k-1}| = |(x^{(N+1)})_{k-1}|$ . Then, dimension  $k-1$  is just rotating after  $N$ .*

If the precondition of Lemma 14 holds, we move to the next dimension  $k-2$ . Otherwise, we want to give a bound such that whenever  $|(x^{(i)})_{k-1}|$  exceeds it, we can conclude that it never decreases back to  $|y_{k-1}|$ . We first introduce the angle  $\gamma(i) = \langle \lambda(x^{(i)})_{k-1} + (x^{(i)})_k, \lambda(x^{(i)})_{k-1} \rangle$ . The angle  $\gamma(i)$  decreases with increasing  $|(x^{(i)})_{k-1}|$ , as dimension  $k$  is just rotating and hence does not change in modulus. We observe that  $\gamma(i) \leq \phi(i)$  for all  $i$ . The following shows that an increase in modulus caused by crossing an “axis” (i.e. if  $\gamma(i) > \pi/2$ ) can only happen once, as in the next step, the angle will have decreased.

► **Lemma 15.** *Let  $a = \lambda(x^{(i)})_{k-1}$  and  $b = (x^{(i)})_k$ . Suppose that  $\theta_g \leq \pi/2$ ,  $\gamma(i) > \pi/2$  and  $|a + b| > |a|$ . Then  $\langle \lambda[a + b], [\lambda b] \rangle \leq \pi/2$ , entailing  $\gamma(i + 1) \leq \phi(i + 1) \leq \pi/2$ .*

Furthermore, a decrease cannot be followed by an increase, unless the angle changes:

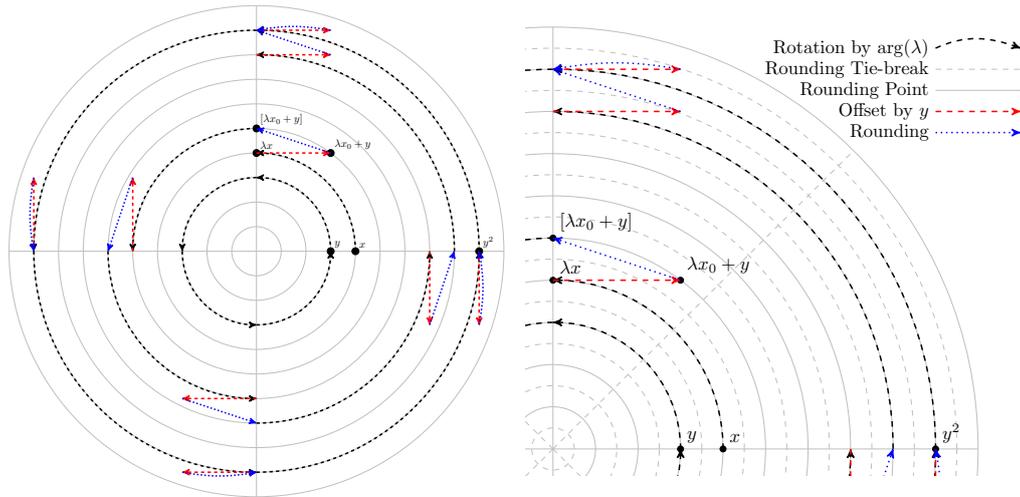
► **Lemma 16.** *Suppose dimension  $k$  is just rotating after  $N$ . It is not possible for  $i - 1 \geq N$ , to have  $\phi(i - 1) = \phi(i) = \phi(i + 1) > \pi/2$  and  $|(x^{(i+1)})_{k-1}| > |(x^{(i)})_{k-1}| < |(x^{(i-1)})_{k-1}|$ .*

Finally, we place a limit on the number of consecutive increases until we can decide that dimension  $k-1$  will not decrease below the current modulus in the future:

► **Lemma 17.** *Let  $a = \lambda(x^{(N)})_{k-1}$  and  $b = (x^{(N)})_k$  for some  $N > 0$ . Suppose that  $k$  is just rotating after  $N$ ,  $|a + b| > |a| + 0.5$  and  $|a| \geq \frac{1}{\sqrt{2}} |b|$ . Then, for all  $i > N$ :  $|(x^{(i)})_{k-1}| > |a|$ .*

With Lemmata 14-17 we are in a position to prove Theorem 12 (the proofs of the preceding lemmata, and the **EXPSpace** analysis can be found in the full version).

**Proof of Theorem 12.** As described above, we consider each dimension separately, starting with  $k = d$ , and assume by induction that the previous dimension is just rotating. We describe an algorithm that tracks the value of  $\phi$  and operates according to Figure 2. Each



■ **Figure 3** Example where the system may become large before being periodic (see Example 18).

realisable value of  $\phi$  relates to a copy of Figure 2 (we only draw one example of  $\phi$  satisfying  $\phi > \pi/2$  and  $\phi \leq \pi/2$  respectively). For  $\phi > \pi/2$  two states are used, one which encodes that the previous transition was decrementing the modulus ( $\phi D$ ), the other which indicates the previous was not decrementing (including first arrival) ( $\phi I$ ).

The algorithm moves on each update step according to the arrow, which denotes whether the update is modulus increasing  $M \uparrow$ , decreasing  $M \downarrow$  or stationary  $MS$ . Similarly  $\phi$  may decrease  $\phi \downarrow$  or stay stationary  $\phi S$ , but never increase (Lemma 13). Whenever  $\phi$  decreases we make progress through the DAG to a lower value of  $\phi$ . All combinations  $\{M \uparrow, M \downarrow, MS\} \times \{\phi \downarrow, \phi S\}$  are accounted for at each state.

Progress is made whenever we move through the DAG towards a stopping criterion. For self-loops a bound is provided (in blue) on the maximum time spent in this state. Since for each dimension we will ultimately end up in just rotating, or be able to stop early, the problem is decidable. ◀

► **Example 18** (System requiring **EXPSPACE** to be periodic). If  $\phi \leq \pi/2$ , the considered dimension will either diverge at some point, or become periodic. This depends, essentially, on whether  $|(x^{(i)})_k| \cos(\phi) < 0.5$ , in which case the rounding will not lead to an increase when  $|(x^{(i)})_{k-1}|$  is sufficiently large relative to  $|(x^{(i)})_k|$ . We give an example where  $|(x^{(i)})_{k-1}|$  grows to  $|(x^{(i)})_k|^2$ , and requires numbers of doubly exponential size (and exponential space) in  $d$  before becoming periodic. We assume that  $\theta_g = \pi/2$  (so there are four possible angles) and integer modulus granularity. Let  $M$  be a single Jordan block of dimension  $d$  with eigenvalue  $\lambda = e^{i\pi/2}$ . The angle  $\phi(i)$  remains constant, but the modulus grows while  $|(x^{(i)})_k| < |(x^{(i)})_{k-1}| \leq |(x^{(i)})_k|^2$ . We start at the point  $x^{(0)} = ((3 + d, 0), \dots, (6, 0), (5, 0), (4, 0))$ , using the representation that  $Ae^{i\theta}$  is written  $(A, \theta)$ . This system is periodic, with maximal component  $x^{(N)} = ((4^{2^{d-1}}, 0), \dots, (4^{2 \cdot 2 \cdot 2}, 0), (4^{2 \cdot 2}, 0), (4^2, 0), (4, 0))$ . Note that  $4^{2^4}$  is larger than a 32-bit number. This idea is illustrated in Figure 3, where  $y$  represents  $(x^{(i)})_k$  and is just rotating, and  $x$  represents  $(x^{(i)})_{k-1}$ , which grows to  $|y|^2$ .

Despite Example 18, which shows that waiting until becoming periodic may need exponential space, we conjecture the Rounded P2P can be solved in **PSPACE**. This is because if  $(x^{(i)})_1$  exceeds a value representable in polynomial space we expect it will never return to the target  $y_1$  (a value representable in polynomial space). However, we are unable to show at the moment that it never gets very large and subsequently returns to a small value.

## 4.2 Argand truncation or expansion in Jordan normal form

We now consider Argand truncation based rounding showing decidability in **EXPSpace**. The rounding function is of the form  $[a + bi] = [a] + [b]i$  where, for  $x \in \mathbb{R}$ ,  $[x] = \lfloor x \rfloor$  if  $x \geq 0$  and  $[x] = \lceil x \rceil$  if  $x < 0$ , which has a non-increasing effect on the modulus.

► **Theorem 19.** *The Rounded P2P Problem is decidable in **EXPSpace** for deterministic Argand rounding function with a non-increasing effect on the modulus and matrices  $M \in \mathbb{A}^{d \times d}$  in Jordan normal form.*

As a key ingredient of Theorem 19 we will make use of the following theorem:

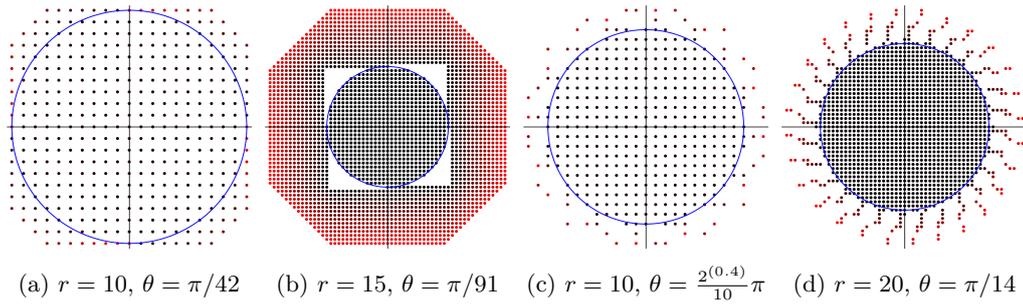
► **Theorem 20** ([28, Corollary 3.12, p.41]). *Both  $x$  is a rational multiple of  $\pi$  and  $\sin(x)$  is rational only at  $\sin(x) = 0, \frac{1}{2},$  or  $1$ . Both  $x$  is a rational multiple of  $\pi$  and  $\cos(x)$  are rational only at  $\cos(x) = 0, \frac{1}{2},$  or  $1$ . Both  $x$  is a rational multiple of  $\pi$  and  $\tan(x)$  are rational only at  $\tan(x) = 0,$  or  $\pm 1$ .*

**Proof sketch of Theorem 19.** Without loss of generality we consider only a single Jordan block with  $|\lambda| = 1$ , as the remaining blocks can be handled in lock step (using the algorithm of Lemma 7 if the eigenvalue is not of modulus one). Consider the  $d$ th component. At each step, whenever rounding takes place, then there is some decrease in the modulus. Thus, either the coordinate hits zero (and stays forever), or it stabilises and becomes periodic (with no rounding ever occurring again). The  $d$ th coordinate can be simulated until this happens. At this point, if its modulus is not  $|y_d|$ ,  $y$  will not be reached in the future and we return NO.

If dimension  $x_d$  reaches zero, then this dimension from some point on becomes irrelevant and the instance can be reduced to an instance of dimension  $d - 1$ . Note that this case must occur if  $\arg(\lambda)$  is not a root of unity as an irrational point is found infinitely often.

In the case where  $x_d$  does not reach zero, then it is periodic at some modulus. This implies it never rounds again, and so surely hits integer points at every step. We show that this can only occur if  $\arg(\lambda)$  is a multiple of  $\pi/2$ . Assume that  $\arg(\lambda)$  is not a multiple of  $\pi/2$ : the rotation of a point with integer coordinate to integer coordinate leads to the conclusion of either rational tangent or rational sine and cosine. By Theorem 20 a rational tangent alongside a rational angle ( $\arg(\lambda)$  is a root of unity) implies that the angle must be a multiple of  $\pi/4$ . It is not  $\pi/4$ , as there is no Pythagorean triangle with angle  $\pi/4$ . By Theorem 20 rational sine and cosine and rational angle concludes the angle must be a multiple of  $\pi/2$ . Finally, we show that when  $\arg(\lambda)$  is a multiple of  $\pi/2$  the system surely diverges at dimension  $d-1$ , and hence we can put a bound on how far we need to simulate. ◀

► **Remark (Argand expansion in Jordan Normal Form).** Instead of considering the rounding function to always decrease the modulus, we consider the rounding function to always increase the modulus. Then, by the same rationality argument either  $\arg(\lambda)$  is a multiple of  $\pi/2$  (so no rounding occurs and standard methods can be applied), or  $\arg(\lambda)$  is not a multiple of  $\pi/2$  and rounding is applied infinitely often. We observe that rounding infinitely often results in divergence. Suppose instead the modulus converges, in supremum, to  $C$ . However the  $[C]$ -ball is finite, thus rounding infinitely often must eventually exhaust the set, contradicting supremacy. Since divergence occurs in the  $d$ th component the system can be iterated until either  $x^{(i)} = y$  or  $(x^{(i)})_d$  exceeds  $y_d$ . (Unless  $(x^{(0)})_d = y_d = 0$ , in which case the  $d$ th component can be deleted.)



■ **Figure 4** Rotational examples. We start with all points in the circle of radius  $r$ , and consider the effect of rotating every point by  $\theta$ , followed by minimal error rounding. This can be seen as viewing the combined orbits, starting at several points. Redder points are added in later generations.

## 5 Discussion of open problems

In this section we consider the following open problem, which already exhibits a technical difficulty for a relatively simple instance.

► **Open Problem 21.** *Under which deterministic bounded-effect rounding functions does the Rounded P2P Problem become decidable (even when restricted to Jordan normal form)?*

In particular we emphasize that even decidability of the Rounded P2P Problem in the case of a 2D *rotation* matrix remains open. This should be compared to the papers [24, 11, 36, 35, 2, 31], which consider linear maps on  $\mathbb{R}^2$  that are close to rotations, and the floor rounding  $\lfloor \cdot \rfloor$  is used to induce discretized maps on  $\mathbb{Z}^2$ . The conjecture made in [35] that all orbits of these maps are eventually periodic (and thus finite) is, to the best of our knowledge, still open in general. This lack of understanding of the dynamics of rotations even on a 2-dimensional lattice is striking and hints at an intrinsic level of difficulty in dealing with eigenvalues of modulus 1.

We ran experiments on the behaviour of rounded orbits induced by rotations in the plane. Four prototypical results are depicted in Figure 4. We note that in every one of our examples the orbits eventually become periodic. Moreover, all experiments fall into the four categories of Figure 4, i.e., where the resulting set consists of (a) a square with cut-off corners, (b) this same square, but with a central square cut out, and (c) all points within the circle with some seemingly randomly added points outside (in the case of an irrational multiple of  $\pi$ ), (d) the initial circle with added “tentacles” occurring in intervals corresponding to the rotational angle (in the case of a rational multiple of  $\pi$ ). We have been unable to construct a rotation with an infinite rounded orbit.

One could hope that other kinds of rounding functions simplify the analysis of the orbits. We have shown truncation based rounding, for example, either helps converge towards zero, or diverge towards infinity, and this can be exploited (particularly at the bottom dimension of a Jordan block). However, roundings which may either round up or down greatly complicate the analysis. Nevertheless, we conjecture that all rounded orbits obtained by rotation eventually become periodic.

**Random rounding functions.** Orbit problems for rounding functions which behave probabilistically are another line of open problems and are a natural candidate for future work.

## References

- 1 Shigeki Akiyama, Tibor Borbély, Horst Brunotte, Attila Pethő, and Jörg Thuswaldner. Generalized radix representations and dynamical systems. I. *Acta Mathematica Hungarica*, 108:207–238, August 2005. doi:10.1007/s10474-005-0221-z.
- 2 Shigeki Akiyama and Attila Pethő. Discretized rotation has infinitely many periodic orbits. *Nonlinearity*, 26(3):871–880, 2013. doi:10.1088/0951-7715/26/3/871.
- 3 S Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Information Processing Letters*, 115(2):155–158, 2015.
- 4 Shaull Almagor, Joël Ouaknine, and James Worrell. The semialgebraic orbit problem. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 6:1–6:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 5 C. Beck and G. Roepstorff. Effects of phase space discretization on the long-time behavior of dynamical systems. *Physica D: Nonlinear Phenomena*, 25(1):173–180, 1987. doi:10.1016/0167-2789(87)90100-X.
- 6 Michael Blank. Ergodic properties of discretizations of dynamic systems. *Dokl. Akad. Nauk SSSR*, 278(4):779–782, 1984.
- 7 Michael Blank. Ergodic properties of a method of numerical simulation of chaotic dynamical systems. *Mathematical Notes of the Academy of Sciences of the USSR*, 45:267–273, 1989. doi:10.1007/BF01158885.
- 8 Michael Blank. Small perturbations of chaotic dynamical systems. *Russian Mathematical Surveys*, 44(6):1–33, December 1989. doi:10.1070/rm1989v044n06abeh002302.
- 9 Michael Blank. Pathologies generated by round-off in dynamical systems. *Physica D: Nonlinear Phenomena*, 78(1):93–114, 1994. doi:10.1016/0167-2789(94)00103-0.
- 10 Michael Blank. *Discreteness and Continuity in Problems of Chaotic Dynamics*. Translations of mathematical monographs. American Mathematical Society, 1997.
- 11 D. Bosio and F. Vivaldi. Round-off errors and  $p$ -adic numbers. *Nonlinearity*, 13(1):309–322, 1999. doi:10.1088/0951-7715/13/1/315.
- 12 Jin-yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *Int. J. Found. Comput. Sci.*, 5(3/4):293–302, 1994. doi:10.1142/S0129054194000165.
- 13 Rebekah Carter and Eva M. Navarro-López. Dynamically-driven timed automaton abstractions for proving liveness of continuous systems. In Marcin Jurdziński and Dejan Ničković, editors, *Formal Modeling and Analysis of Timed Systems*, pages 59–74, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33365-1\_6.
- 14 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The polyhedron-hitting problem. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 940–956. SIAM, 2015.
- 15 Eva Darulova, Anastasiia Izycheva, Fariha Nasir, Fabian Ritter, Heiko Becker, and Robert Bastian. Daisy - framework for analysis and optimization of numerical programs (tool paper). In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 270–287, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89960-2\_15.
- 16 Phil Diamond and Igor Vladimirov. Asymptotic independence and uniform distribution of quantization errors for spatially discretized dynamical systems. *International Journal of Bifurcation and Chaos*, 8:1479–1490, 1998. doi:10.1142/S0218127498001133.
- 17 Phil Diamond and Igor Vladimirov. Set-valued Markov chains and negative semitrajectories of discretized dynamical systems. *Journal of Nonlinear Science*, 12:113–141, 2002. doi:10.1007/s00332-001-0450-4.
- 18 S.P. Dias, L. Longa, and E. Curado. Influence of the finite precision on the simulations of discrete dynamical systems. *Communications in Nonlinear Science and Numerical Simulation*, 16(3):1574–1579, 2011. doi:10.1016/j.cnsns.2010.07.003.

- 19 Eric Goubault and Sylvie Putot. Static analysis of finite precision computations. In Ranjit Jhala and David Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 232–247, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-18275-4\_17.
- 20 Stephen Hammel, James Yorke, and Celso Grebogi. Numerical orbits of chaotic processes represent true orbits. *Bull. Amer. Math. Soc.*, 19:465–, April 1988. doi:10.1090/S0273-0979-1988-15701-1.
- 21 Anastasiia Izycheva and Eva Darulova. On sound relative error bounds for floating-point arithmetic. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, FMCAD '17, page 15–22, Austin, Texas, 2017. FMCAD Inc. doi:10.23919/FMCAD.2017.8102236.
- 22 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, 1986. doi:10.1145/6490.6496.
- 23 Anatole Katok and Boris Hasselblatt. *Introduction to the Modern Theory of Dynamical Systems*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CB09780511809187.
- 24 John Lowenstein, Spyros Hatjispyros, and Franco Vivaldi. Quasi-periodicity, global stability and scaling in a model of Hamiltonian round-off. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 7(1):49–66, 1997. doi:10.1063/1.166240.
- 25 Victor Magron, George Constantinides, and Alastair Donaldson. Certified roundoff error bounds using semidefinite programming. *ACM Trans. Math. Softw.*, 43(4), 2017. doi:10.1145/3015465.
- 26 Oded Maler and Grégory Batt. Approximating continuous systems by timed automata. In Jasmin Fisher, editor, *Formal Methods in Systems Biology*, pages 77–89, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-68413-8\_6.
- 27 Mariano Moscato, Laura Titolo, Aaron Dutle, and César A. Muñoz. Automatic estimation of verified floating-point round-off errors via static analysis. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 213–229, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-66266-4\_14.
- 28 Ivan Niven. *Irrational Numbers*. Number 11 in The Carus Mathematical Monographs. The Mathematical Association of America, 1956. doi:10.5948/9781614440116.
- 29 Helena E. Nusse and James A. Yorke. Is every approximate trajectory of some process near an exact trajectory of a nearby process? *Comm. Math. Phys.*, 114(3):363–379, 1988. doi:10.1007/BF01242136.
- 30 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, 2015.
- 31 Attila Pethö, Jörg M. Thuswaldner, and Mario Weitzer. The finiteness property for shift radix systems with general parameters. *Integers*, 19:A50, 2019. URL: <http://math.colgate.edu/%7Eintegers/t50/t50.Abstract.html>.
- 32 Stefano Schivo and Rom Langerak. Discretization of continuous dynamical systems using UPPAAL. In Joost-Pieter Katoen, Rom Langerak, and Arend Rensink, editors, *ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, volume 10500 of *Lecture Notes in Computer Science*, pages 297–315. Springer, 2017. doi:10.1007/978-3-319-68270-9\_15.
- 33 Alexey Solovyev, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. Rigorous estimation of floating-point round-off errors with symbolic Taylor expansions. In Nikolaj Bjørner and Frank de Boer, editors, *FM 2015: Formal Methods*, pages 532–550, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-19249-9\_33.
- 34 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the*

- 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.
- 35 Franco Vivaldi. The arithmetic of discretized rotations. *AIP Conference Proceedings*, 826, March 2006. doi:10.1063/1.2193120.
- 36 Franco Vivaldi and Igor Vladimirov. Pseudo-randomness of round-off errors in discretized linear maps on the plane. *International Journal of Bifurcation and Chaos*, 13(11):3373–3393, 2003. doi:10.1142/S0218127403008557.



# Parameterized Complexity of Safety of Threshold Automata

A. R. Balasubramanian

Technische Universität München, Germany

bala.ayikudi@tum.de

---

## Abstract

Threshold automata are a formalism for modeling fault-tolerant distributed algorithms. In this paper, we study the parameterized complexity of reachability of threshold automata. As a first result, we show that the problem becomes  $W[1]$ -hard even when parameterized by parameters which are quite small in practice. We then consider two restricted cases which arise in practice and provide fixed-parameter tractable algorithms for both these cases. Finally, we report on experimental results conducted on some protocols taken from the literature.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models; Theory of computation → Logic and verification

**Keywords and phrases** Threshold automata, distributed algorithms, parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.37

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

**Acknowledgements** I would like to thank Prof. Javier Esparza for useful discussions regarding the paper and Christoph Welzel and Margarete Richter for their encouragement and support. I would also like to thank the anonymous reviewers whose comments greatly improved the presentation of the paper.

## 1 Introduction

Threshold automata [21] are a formalism for modeling and analyzing fault-tolerant distributed algorithms. In this setup, an arbitrary but fixed number of processes execute a given protocol as specified by a threshold automaton. Verification of these systems aims to prove these protocols correct for any number of processes [4].

One of the main differences between threshold automata and other formalisms for modeling distributed protocols (like replicated systems and population protocols [1, 16, 18]) is the notion of a *threshold guard*. Roughly speaking, a threshold guard specifies certain constraints that should hold between the number of messages received and the number of participating processes, in order for a transition to be enabled. For example, a guard of the form  $x \geq n/2 - t$ , (where  $x$  counts the number of messages of some specified type,  $n$  is the number of processes and  $t$  is the maximum number of faulty processes), specifies that the number of messages received should be bigger than  $n/2 - t$ , in order for the process to proceed. This feature is important in modeling fault-tolerant distributed algorithms where, often a process can make a move only if it has received a message from a majority or two-thirds of the number of processes. In a collection of papers [26, 3, 25, 24, 23], many algorithms have been developed for verifying various properties for threshold automata. These algorithms have then been used to verify a number of protocols from the distributed computing literature [24]. It is known that the reachability problem for threshold automata is NP-complete [2].



© A. R. Balasubramanian;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 37; pp. 37:1–37:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

*Parameterized complexity* [13] attempts to study decision problems that come along with a *parameter*. In parameterized complexity, apart from the size of the input  $n$ , one considers further parameters  $k$  that capture the structure of the input and one looks for algorithms that run in time  $f(k) \cdot n^{O(1)}$ , where  $f$  is some function dependent on  $k$  alone. The hope is to find parameters which are quite small in practice and to base the dominant running time of the algorithm on this parameter alone. Problems solvable in such a manner are called fixed-parameter tractable (FPT).

In recent years, increasing effort has been devoted to studying the parameterized complexity of problems in verification for different models [10, 11, 15, 17, 9]. Motivated by this and by the hard theoretical complexity (NP-completeness) of reachability of threshold automata, we study the parameterized complexity of the same problem, parameterized by (among other parameters) the number of threshold guards and the given safety specification. Our first result is a parameterized equivalent of NP-hardness, which shows that reachability of threshold automata is W[1]-hard. However, motivated by practical concerns, we then study two restricted cases for which we provide fixed-parameter tractable algorithms. In the first case, we restrict ourselves to threshold automata whose underlying control structure is acyclic and provide a simple algorithm which reduces the size of the automaton to a function of the considered parameters. In the second case we consider multiplicative threshold automata where the number of *fall* guards is a constant. (For a definition of fall guards, see Section 2.1.) Roughly speaking, multiplicativity means that any run over a smaller population of processes can be “lifted” to a run over a bigger population as well. For this case, we use results from Petri net theory to provide an FPT algorithm. Finally, the usefulness of these cases is shown by a preliminary implementation of our algorithms on various protocols from the benchmark in [24]. Our implementation compares favorably with ByMC, the tool developed in [24] and also with the algorithm given in [2].

## 2 Preliminaries

We denote the set of non-negative integers by  $\mathbb{N}_0$ , the set of positive integers by  $\mathbb{N}_{>0}$  and the set of all non-negative rational numbers by  $\mathbb{Q}_{\geq 0}$ .

### 2.1 Threshold Automata

We introduce threshold automata, mostly following the definition and notations used in [2]. Along the way, we also illustrate the definitions on the example of Figure 2 from [25], which is a model of the Byzantine agreement protocol of Figure 1.

#### Environments

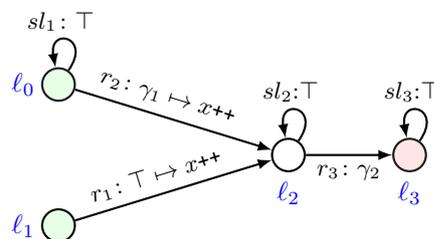
Threshold automata are defined relative to an *environment*  $Env = (\Pi, RC, Num)$ , where  $\Pi$  is a set of *environment variables* ranging over  $\mathbb{N}_0$ ,  $RC \subseteq \mathbb{N}_0^\Pi$  is a *resilience condition* over the environment variables, expressible as a linear formula, and  $Num: RC \rightarrow \mathbb{N}_0$  is a linear function called the *number function*. Intuitively, a valuation of  $\Pi$  determines the number of processes of different kinds (e.g. faulty) executing the protocol, and  $RC$  describes the admissible combinations of values of environment variables. Finally,  $Num$  associates to a each admissible combination, the number of copies of the automaton that are going to run in parallel, or, equivalently, the number of processes explicitly modeled. In a Byzantine setting, faulty processes behave arbitrarily, and so we do not model them explicitly; in this case, the system consists of one copy of the automaton for every correct process. In the crash fault model, processes behave correctly until they crash and they must be modeled explicitly.

```

1 var  $myval_i \in \{0, 1\}$ 
2 var  $accept_i \in \{false, true\} \leftarrow false$ 
3
4 while true do (in one atomic step)
5   if  $myval_i = 1$ 
6     and not sent ECHO before
7     then send ECHO to all
8
9   if received ECHO from at least
10     $t + 1$  distinct processes
11    and not sent ECHO before
12    then send ECHO to all
13
14  if received ECHO from at least
15    $n - t$  distinct processes
16  then  $accept_i \leftarrow true$ 
17 od

```

■ **Figure 1** Pseudocode of a reliable broadcast protocol from [28] for a correct process  $i$ , where  $n$  and  $t$  denote the number of processes, and an upper bound on the number of faulty processes. The protocol satisfies its specification (if  $myval_i = 0$  for every correct process  $i$ , then no correct process sets its  $accept$  variable to  $true$ ) if  $t < n/3$ .



■ **Figure 2** Threshold automaton from [25] modeling the body of the loop in the protocol from Fig. 1. Symbols  $\gamma_1, \gamma_2$  stand for the threshold guards  $x \geq (t + 1) - f$  and  $x \geq (n - t) - f$ , where  $n$  and  $t$  are as in Fig. 1, and  $f$  is the actual number of faulty processes. The shared variable  $x$  models the number of ECHO messages sent by correct processes. Processes with  $myval_i = b$  (line 1) start in location  $\ell_b$  (in green). Rules  $r_1$  and  $r_2$  model sending ECHO at lines 7 and 12. The self-loop rules  $sl_1, \dots, sl_3$  are stuttering steps.

► **Example 1.** In the threshold automaton of Figure 2, the environment variables are  $n$ ,  $f$ , and  $t$ , describing the number of processes, the number of (Byzantine) faulty processes, and the maximum possible number of faulty processes, respectively. The resilience condition is the constraint  $n/3 > t \geq f$ . The function  $Num$  is given by  $Num(n, t, f) = n - f$ , which is the number of correct processes.

## Threshold automata

A *threshold automaton* over an environment  $Env$  is a tuple  $\text{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R})$ , where  $\mathcal{L}$  is a finite set of *local states* (or *locations*),  $\mathcal{I} \subseteq \mathcal{L}$  is a nonempty subset of *initial locations*,  $\Gamma$  is a set of *shared variables* ranging over  $\mathbb{N}_0$ , and  $\mathcal{R}$  is a set of *transition rules* (or *just rules*), formally described below.

A *transition rule* (or just a *rule*) is a tuple  $r = (\text{from}, \text{to}, \varphi, \vec{u})$ , where *from* and *to* are the *source* and *target* locations,  $\varphi \subseteq \mathbb{N}_0^{\Pi \cup \Gamma}$  is a conjunction of *threshold guards* (described below), and  $\vec{u}: \Gamma \rightarrow \{0, 1\}$  is an *update*. We often let  $r.\text{from}, r.\text{to}, r.\varphi, r.\vec{u}$  denote the components of  $r$ . Intuitively,  $r$  states that a process can move from *from* to *to* if the current values of  $\Pi$  and  $\Gamma$  satisfy  $\varphi$ , and when it moves, it updates the current valuation  $\vec{g}$  of  $\Gamma$  by performing the update  $\vec{g} := \vec{g} + \vec{u}$ . Since all components of  $\vec{u}$  are nonnegative, the values of shared variables never decrease. A *threshold guard*  $\varphi$  has one of the following forms:  $b \cdot x \bowtie a_0 + a_1 \cdot p_1 + \dots + a_{|\Pi|} \cdot p_{|\Pi|}$  where  $\bowtie \in \{\geq, <\}$ ,  $x \in \Gamma$  is a shared variable,  $p_1, \dots, p_{|\Pi|} \in \Pi$  are the environment variables,  $b \in \mathbb{N}_{>0}$  and  $a_0, a_1, \dots, a_{|\Pi|} \in \mathbb{Z}$  are integer coefficients. If  $b = 1$ , then the guard is called a *simple guard*. Additionally, if  $\bowtie = \geq$ , then the guard is called a *rise guard* and otherwise the guard is called a *fall guard*. We sometimes use  $b \cdot x = a_0 + a_1 \cdot p_1 + \dots + a_{|\Pi|} \cdot p_{|\Pi|}$  as a shorthand for  $b \cdot x \geq a_0 + a_1 \cdot p_1 + \dots + a_{|\Pi|} \cdot p_{|\Pi|} \wedge b \cdot x < (a_0 + 1) + a_1 \cdot p_1 + \dots + a_{|\Pi|} \cdot p_{|\Pi|}$ . Since

shared variables are initialized to 0 and they never decrease, once a rise (resp. fall) guard becomes true (resp. false) it stays true (resp. false). We call this property *monotonicity of guards*. We let  $\Phi^{\text{rise}}$ ,  $\Phi^{\text{fall}}$ , and  $\Phi$  denote the sets of rise guards, fall guards, and all guards of TA. Finally, the *graph* of TA is the graph where the vertices are the locations and there is an edge between  $\ell$  and  $\ell'$  if there is a rule  $r$  in TA with  $r.\text{from} = \ell$  and  $r.\text{to} = \ell'$ . We say that TA is acyclic if its graph is acyclic.

► **Example 2.** The rule  $r_2$  of Figure 2 has  $\ell_0$  and  $\ell_2$  as source and target locations,  $x \geq (t+1) - f$  as guard, and increments the value of the shared variable  $x$  by 1.

## Configurations and transition relation

A *configuration* of TA is a triple  $\sigma = (\vec{\kappa}, \vec{g}, \vec{p})$  where  $\vec{\kappa}: \mathcal{L} \rightarrow \mathbb{N}_0$  describes the number of processes at each location, and  $\vec{g} \in \mathbb{N}_0^{|\Gamma|}$  and  $\vec{p} \in RC$  are valuations of the shared variables and the environment variables. In particular,  $\sum_{\ell \in \mathcal{L}} \vec{\kappa}(\ell) = \text{Num}(\vec{p})$  always holds. A configuration is *initial* if  $\vec{\kappa}(\ell) = 0$  for every  $\ell \notin \mathcal{I}$ , and  $\vec{g} = \vec{0}$ . We often let  $\sigma.\vec{\kappa}, \sigma.\vec{g}, \sigma.\vec{p}$  denote the components of  $\sigma$ .

A configuration  $\sigma = (\vec{\kappa}, \vec{g}, \vec{p})$  *enables* a rule  $r = (\text{from}, \text{to}, \varphi, \vec{u})$  if  $\vec{\kappa}(\text{from}) > 0$ , and  $(\vec{g}, \vec{p})$  satisfies the guard  $\varphi$ , i.e., substituting  $\vec{g}(x)$  for  $x$  and  $\vec{p}(p_i)$  for  $p_i$  in  $\varphi$  yields a true expression, denoted by  $\sigma \models \varphi$ . If  $\sigma$  enables  $r$ , then TA can *move* from  $\sigma$  to the configuration  $r(\sigma) = (\vec{\kappa}', \vec{g}', \vec{p}')$  defined as follows: (i)  $\vec{p}' = \vec{p}$ , (ii)  $\vec{g}' = \vec{g} + \vec{u}$ , and (iii)  $\vec{\kappa}' = \vec{\kappa} + \vec{v}_r$ , where  $\vec{v}_r = \vec{0}$  if  $\text{from} = \text{to}$  and otherwise,  $\vec{v}_r(\text{from}) = -1$ ,  $\vec{v}_r(\text{to}) = +1$ , and  $\vec{v}_r(\ell) = 0$  for all other locations  $\ell$ . We let  $\sigma \rightarrow r(\sigma)$  denote that TA can move from  $\sigma$  to  $r(\sigma)$ .

## Schedules and paths

A *schedule* is a (finite or infinite) sequence of rules. A schedule  $\tau = r_1, \dots, r_m$  is *applicable* to configuration  $\sigma_0$  if there is a sequence of configurations  $\sigma_1, \dots, \sigma_m$  such that  $\sigma_i = r_i(\sigma_{i-1})$  for  $1 \leq i \leq m$ , and we define  $\tau(\sigma_0) := \sigma_m$ . We let  $\sigma \xrightarrow{*} \sigma'$  denote that  $\tau(\sigma) = \sigma'$  for some schedule  $\tau$ , and say that  $\sigma'$  is *reachable* from  $\sigma$ . Further we let  $\tau \cdot \tau'$  denote the concatenation of two schedules  $\tau$  and  $\tau'$ , and, given  $\mu \geq 0$ , let  $\mu \cdot \tau$  the concatenation of  $\tau$  with itself  $\mu$  times.

A *path* or *run* is a finite or infinite sequence  $\sigma_0, r_1, \sigma_1, \dots, \sigma_{k-1}, r_k, \sigma_k, \dots$  of alternating configurations and rules such that  $\sigma_i = r_i(\sigma_{i-1})$  for every  $r_i$  in the sequence. If  $\tau = r_1, \dots, r_{|\tau|}$  is applicable to  $\sigma_0$ , then we let  $\text{path}(\sigma_0, \tau)$  denote the path  $\sigma_0, r_1, \sigma_1, \dots, r_{|\tau|}, \sigma_{|\tau|}$  with  $\sigma_i = r_i(\sigma_{i-1})$ , for  $1 \leq i \leq |\tau|$ . Similarly, if  $\tau$  is an infinite schedule. Given a path  $\text{path}(\sigma, \tau)$ , the set of all configurations in the path is denoted by  $\text{Cfgs}(\sigma, \tau)$ .

The main focus of this paper will be the *reachability problem* and is defined as: Given TA and a set of locations  $\mathcal{L}_{\text{spec}} = \mathcal{L}_{=0} \cup \mathcal{L}_{>0}$  (called the specification), decide if there is a run of TA satisfying  $\mathcal{L}_{\text{spec}}$ , i.e., decide if there is an initial configuration  $\sigma_0$  such that some  $\sigma$  reachable from  $\sigma_0$  satisfies  $\sigma.\vec{\kappa}(\ell) = 0$  for every  $\ell \in \mathcal{L}_{=0}$  and  $\sigma.\vec{\kappa}(\ell) > 0$  for every  $\ell \in \mathcal{L}_{>0}$ . The *coverability problem* is the special case of the reachability problem where  $\mathcal{L}_{=0} = \emptyset$ .

## 2.2 Fixed-parameter tractability

We refer the reader to [13] for more information on parameterized complexity and only give the necessary definitions here. A *parameterized problem*  $L$  is a subset of  $\Sigma^* \times \mathbb{N}_0$  for some alphabet  $\Sigma$ . A parameterized problem  $L$  is said to be *fixed-parameter tractable* (FPT) if there exists an algorithm  $A$  such that  $(x, k) \in L$  iff  $A(x, k)$  is true and  $A$  runs in time  $f(k) \cdot |x|^{O(1)}$  for some computable function  $f$ , depending only on the *parameter*  $k$ . Given

parameterized problems  $L, L' \subseteq \Sigma^* \times \mathbb{N}_0$  we say that  $L$  is reducible to  $L'$  if there is an algorithm that, given an input  $(x, k)$ , produces another input  $(x', k')$  in time  $f(k) \cdot |x|^{O(1)}$  such that  $(x, k) \in L \iff (x', k') \in L'$  and  $k' \leq g(k)$  for some functions  $f$  and  $g$  depending only on  $k$ .

The parameterized clique problem is the set of all pairs  $(G, k)$  such that the graph  $G$  has a clique of size  $k$ . A parameterized problem  $L$  is said to be  $W[1]$ -hard if there is a parameterized reduction from  $L$  to the parameterized clique problem. If  $L$  is  $W[1]$ -hard and there is a parameterized reduction from  $L$  to  $L'$  then  $L'$  is  $W[1]$ -hard as well.  $W[1]$ -hardness is usually taken to be evidence that the problem does not have an FPT algorithm.

### 3 $W[1]$ -hardness

We consider the reachability problem parameterized by the following parameters:  $|\Phi|$  (the number of distinct guards),  $|\mathcal{L}_{\text{spec}}|$  (the size of the specification),  $|RC|$  (the number of constraints in the resilience condition) and  $C$  (the maximum constant appearing in any of the guards of TA). (We note that if  $x \in \Gamma \cup \Pi$  such that  $x$  does not appear in any of the guards in  $\Phi$  or in any of the constraints in  $RC$ , then  $x$  can be removed from the input. Hence, we will always assume that  $|\Gamma| + |\Pi| \leq |\Phi| + |RC|$  and for this reason, we do not consider  $|\Gamma|$  and  $|\Pi|$  explicitly as parameters.) In practice, all these values are quite small, roughly in the range of 10 to 25. Unfortunately, we prove the following negative result:

► **Theorem 3.** *Coverability (and hence reachability) for threshold automata parameterized by  $|\Phi| + |\mathcal{L}_{\text{spec}}| + |RC| + C$  is  $W[1]$ -hard, even for acyclic automata where  $|\Phi^{\text{fall}}|$  is a constant.*

**Proof.** We give a parameterized reduction from the *Unary Bin Packing* problem which is known to be  $W[1]$ -hard (See Theorem 2 of [20]) and is defined as follows:

*Given:* A finite set of items  $I = \{0, 1, 2, \dots, w\}$ , a size  $\text{size}(i) \in \mathbb{N}_0$  for each  $i \in I$ , two positive integers  $B$  and  $k$ . (The integers  $\text{size}(i)$  and  $B$  are encoded in unary)

*Parameter:*  $k$

*Decide:* If there exists a partition of  $I$  into bins  $I_1, \dots, I_k$  such that the sum of the sizes of the items in each bin  $I_j$  is less than or equal to  $B$

Let  $\text{size} = \sum_{i \in I} \text{size}(i)$ . Our parameterized reduction works as follows: We will have  $k + 1$  environment variables  $c_1, c_2, \dots, c_k, n$ . Intuitively  $c_i$  will denote the sum of the sizes of the items in the  $i^{\text{th}}$  bin. The environment variable  $n$  will denote the number of processes modeled.

Further, we will have  $k + 5$  shared variables  $x_1, \dots, x_k, \text{access}_1, \text{access}_2, \text{access}_3$  and  $\text{count}_1, \text{count}_2$ . The variable  $x_i$  will denote the sum of the sizes of items which **do not belong** to the  $i^{\text{th}}$  bin. The role of  $\text{count}_1$  and  $\text{count}_2$  will be to set up two counters whose value will be exactly  $\text{size}$  and  $B$  respectively. Our construction will have three gadgets and the role of  $\text{access}_1, \text{access}_2$  and  $\text{access}_3$  is to ensure that exactly one process can enter the first, second and third gadgets respectively.

We will have exactly one initial location *start* and three rules of the form  $r_1 : (\text{start}, \text{access}_1 < 1, \text{access}_1++, p_0)$ ,  $r_2 : (\text{start}, \text{access}_2 < 1, \text{access}_2++, q_0)$  and  $r_3 : (\text{start}, \text{access}_3 < 1, \text{access}_3++, \ell_0)$ . This means that once a process fires  $r_1$ , it increments  $\text{access}_1$  and hence no other process can fire  $r_1$  in the future. Similarly for the rules  $r_2$  and  $r_3$ . Hence these three rules ensure that at most one process can enter  $p_0, q_0$  and  $\ell_0$  respectively. For the specification, we set  $\mathcal{L}_{=0} = \emptyset$  and  $\mathcal{L}_{>0} = \{p_{\text{corr}}, q_{\text{corr}}, \ell_{w+1}\}$ , whose locations we will now explain.



■ **Figure 3** First gadget, which sets up the value of  $count_1$  to be exactly  $size$ .



■ **Figure 4** Second gadget, which sets up the value of  $count_2$  to be exactly  $B$ .



■ **Figure 5** Third gadget, which guesses the partition. Here  $cond_j$  is the condition  $x_j \geq \sum_{l \neq j} c_l$  and  $upd_j$  is the update  $\bigwedge_{l \neq j} x_l++$ .

The first gadget is given by Figure 3 and starts from the location  $p_0$ . It increments the shared variable  $count_1$  to the value  $size$ . This gadget then ensures that we can reach  $p_{corr}$  only if the sum of the values of the environment variables  $c_1, c_2, \dots, c_k$  is exactly  $size$ . Notice that this gadget can be constructed in polynomial time, since each  $size(i)$  is given in unary.

The second gadget is given by Figure 4 and starts from the location  $q_0$ . It increments the shared variable  $count_2$  to the value  $B$ . This gadget then ensures that we can reach  $q_{corr}$  only if the values of the environment variables  $c_1, c_2, \dots, c_k$  are all at most  $B$ . Notice that this gadget can be constructed in polynomial time, since  $B$  is given in unary.

The third gadget is comprised of locations  $\{\ell_i\}_{0 \leq i \leq w+1}$  and  $\{\ell_{i,q}^j\}_{0 \leq i \leq w, 0 \leq q \leq size(i), 1 \leq j \leq k}$  and is comprised of various mini-gadgets. For every  $0 \leq i \leq w$  and  $1 \leq j \leq k$ , the third gadget has a mini-gadget as given by Figure 5.

Recall that the shared variable  $x_j$  denotes the the sum of the sizes of items which **do not** belong to the  $j^{th}$  bin. Intuitively, if a process moves from  $\ell_i$  to  $\ell_{i+1}$  by going through  $\ell_{i,0}^j, \dots, \ell_{i,size(i)}^j$ , this corresponds to putting the  $i^{th}$  item in the  $j^{th}$  bin and hence the mini-gadget increments the variables  $\{x_l\}_{l \neq j}$  by the value  $size(i)$ . To ensure that we do not overshoot the bin size of the  $j^{th}$  bin, we have the guards  $x_j \geq \sum_{l \neq j} c_l$  at the beginning and the end of the mini-gadget. Recall that the first gadget ensures that  $\sum_{1 \leq l \leq k} c_l = size$  and since  $x_j$  denotes the sum of sizes of items **not in** the  $j^{th}$  bin, the condition  $x_j \geq \sum_{l \neq j} c_l$  ensures that the sum of the sizes of the items in the  $j^{th}$  bin is at most  $c_j$ . Since the second gadget forces  $c_j \leq B$ , it follows that the test  $x_j \geq \sum_{l \neq j} c_l$  ensures that the sum of the sizes **in** the  $j^{th}$  bin is at most  $B$ . Notice that, once again this gadget can be constructed in polynomial time, since each  $size(i)$  is given in unary.

Let  $RC$  be  $n > 1$  and let  $Num(c_1, \dots, c_k, n) = n$ . From the given construction it is clear that a configuration satisfying  $\mathcal{L}_{spec}$  is reachable iff we can partition  $I$  into  $k$  bins such that the sum of sizes of items in each bin does not exceed  $B$ .

It is clear that the reduction can be accomplished in polynomial time. Notice that the automaton is acyclic,  $\mathcal{L}_{=0} = \emptyset$ ,  $|\Phi| = O(k)$ ,  $|\Phi^{fall}| = 4$ ,  $|RC| = 1$ ,  $C = 1$  and  $|\mathcal{L}_{spec}| = 3$ . Hence it is clear that  $|\Phi| + |RC| + C + |\mathcal{L}_{spec}| = O(k)$  and so the above reduction is indeed a parameterized reduction from the unary bin packing problem to the coverability problem. ◀

We now identify two special cases for which we give an FPT algorithm and discuss how these special cases arise in practice for a variety of distributed algorithms.

#### 4 Acyclic threshold automata

The first case we consider is that of acyclic threshold automata, i.e., threshold automata whose underlying graph is acyclic. Except for one protocol, all the others in the benchmark of [24] are acyclic.<sup>1</sup> As the reduction of Theorem 3 produces acyclic threshold automata, we cannot hope for an FPT algorithm parameterized by  $\{|\Phi|, |\mathcal{L}_{\text{spec}}|, |RC|, C\}$ . However, we show that

► **Theorem 4.** *Reachability of acyclic threshold automata parameterized by  $|\Phi| + |\mathcal{L}_{\text{spec}}| + |RC| + C + D$  is in FPT, where  $D$  is the length of the longest path in the graph of the threshold automaton.*

**Proof.** Let TA be the given acyclic threshold automaton. First, we show that it is possible to incrementally “contract” the locations of TA in a bottom-up manner, while preserving the reachability property, such that, in the resulting automaton after contraction, the number of locations and rules is a function of  $|\Phi| + |\mathcal{L}_{\text{spec}}| + |RC| + C + D$ . This then immediately implies our theorem, since the size of the whole automaton is now just a function of  $|\Phi| + |\mathcal{L}_{\text{spec}}| + |RC| + C + D$ .

More formally, let the contraction of a subset  $S = \{\ell_1, \dots, \ell_q\}$  of locations of TA be the following operation: We remove the locations  $\ell_1, \dots, \ell_q$  from TA, introduce a new location  $\ell_S$  and we replace all occurrences of  $\ell_1, \dots, \ell_q$  in every rule of TA with  $\ell_S$ . We say that a set  $S$  in TA is *good* if for every two locations  $\ell, \ell' \in S$ , if  $(\ell, \ell'', \phi, \vec{u})$  is a rule in TA then  $(\ell', \ell'', \phi, \vec{u})$  is also a rule in TA. Intuitively, this means that, for every rule that we can fire from  $\ell$ , there is another rule we can fire from  $\ell'$  which will have the exact same effect. Since TA is assumed to be acyclic, contracting a good set cannot introduce cycles. Let  $\text{Tar} = \{\ell : \ell \in \mathcal{L}_{\text{spec}}\}$ . The following is a very simple fact to verify:

*Claim:* Suppose  $S$  is a good set such that  $S \cap \text{Tar} = \emptyset$  and let  $\text{TA}'$  be the threshold automaton obtained by contracting  $S$  in TA. Then TA satisfies  $\mathcal{L}_{\text{spec}}$  iff  $\text{TA}'$  does.

Given a threshold automaton TA such that  $D$  is the length of the longest path in its graph, the “layers” of TA is a partition of the locations into subsets  $L_0^{\text{TA}}, L_1^{\text{TA}}, \dots, L_D^{\text{TA}}$  such that  $\ell \in L_i^{\text{TA}}$  iff the longest path ending at  $\ell$  in the graph of TA is of length  $i$ . The subset  $L_i^{\text{TA}}$  will be called the  $i^{\text{th}}$  layer of TA. We will now construct a sequence of threshold automata  $\text{TA}_D, \text{TA}_{D-1}, \dots, \text{TA}_0$  such that for each  $i$ ,  $|L_i^{\text{TA}_i}| + |L_{i+1}^{\text{TA}_i}| + \dots + |L_D^{\text{TA}_i}| \leq g_i(|\Phi|, |RC|, |\mathcal{L}_{\text{spec}}|, D)$  for some function  $g_i$  and such that  $\text{TA}_i$  satisfies  $\mathcal{L}_{\text{spec}}$  iff  $\text{TA}_{i+1}$  does.

For the base case of  $\text{TA}_D$ , we take the threshold automaton TA and consider the set  $S_D := L_D^{\text{TA}} \setminus \text{Tar}$ . We now contract  $S_D$  in TA to get a threshold automaton  $\text{TA}_D$ . Notice that  $S_D$  is a good set and by the above claim,  $\text{TA}_D$  satisfies  $\mathcal{L}_{\text{spec}}$  iff TA does.

For the induction step, suppose we have already constructed  $\text{TA}_{i+1}$ . For a location  $\ell \in L_i^{\text{TA}_{i+1}}$ , define its *color* to be the set  $\{(\ell', \phi, \vec{u}) : (\ell, \ell', \phi, \vec{u}) \text{ is a rule in } \text{TA}_{i+1}\}$ . Observe that if  $\ell \in L_i^{\text{TA}_{i+1}}$  and  $(\ell, \ell', \phi, \vec{u})$  is a rule in  $\text{TA}_{i+1}$  then  $\ell' \in L_{i+1}^{\text{TA}_{i+1}} \cup L_{i+2}^{\text{TA}_{i+1}} \cup \dots \cup L_D^{\text{TA}_{i+1}}$ . By induction hypothesis,  $|L_{i+1}^{\text{TA}_{i+1}} \cup L_{i+2}^{\text{TA}_{i+1}} \cup \dots \cup L_D^{\text{TA}_{i+1}}| \leq g_{i+1}(|\Phi|, |RC|, |\mathcal{L}_{\text{spec}}|, D)$  for

<sup>1</sup> Some of the examples have self-loops on some locations, but since these self-loops do not update any of the shared variables, we can remove them without affecting the reachability relation.

some function  $g_{i+1}$ . It then follows that the number of possible colors is at most  $2^{|\Phi|} \cdot 2^{|\Gamma|} \cdot g_{i+1}(|\Phi|, |\mathcal{L}_{\text{spec}}|, D)$ . Hence as long as the number of locations in  $L_i^{\text{TA}_{i+1}}$  is bigger than  $2^{|\Phi|} \cdot 2^{|\Gamma|} \cdot g_{i+1}(|\Phi|, |\mathcal{L}_{\text{spec}}|, D) + |\text{Tar}|$  there will be two locations in  $L_i^{\text{TA}_{i+1}} \setminus \text{Tar}$  which have the same color and can hence be contracted while maintaining the answer for  $\mathcal{L}_{\text{spec}}$ . It then follows that by repeated contraction, we can finally end up at a threshold automaton  $\text{TA}_i$  such that  $|L_i^{\text{TA}_i}| + \dots + |L_D^{\text{TA}_i}| \leq O(2^{|\Phi|} \cdot 2^{|\Phi|+|RC|} \cdot g_{i+1}(|\Phi|, |RC|, |\mathcal{L}_{\text{spec}}|, D) + |\text{Tar}|)$ . Taking this bound to be the function  $g_i$ , we get our required  $\text{TA}_i$ .<sup>2</sup>

Notice that the number of locations (and also rules) in  $\text{TA}_0$  is only dependent on  $|\Phi|, |RC|, |\mathcal{L}_{\text{spec}}|$  and  $D$ . Since the reachability problem is decidable, it immediately follows that we have a parameterized algorithm for acyclic threshold automata running in time  $f(|\Phi| + |RC| + |\mathcal{L}_{\text{spec}}| + C + D) \cdot n^{O(1)}$  ◀

## 5 Threshold automata with constantly many fall guards

As a second case, we consider threshold automata in which the number of fall guards is a constant. In almost all of the benchmarks of [24], the number of fall guards is at most one. We provide some intuitive reason behind this phenomenon. In threshold automata, shared variables are usually used for two things: To record that some process has sent a message or to keep track of the number of processes which have crashed so far. If a shared variable  $v$  is used for the first purpose, then all guards containing  $v$  are typically rise guards, since we only want to check that enough messages have been received to proceed. On the other hand, if  $v$  is used to keep track of the number of crashed processes, then we will have a fall guard which allows a process to crash only if the value of  $v$  is less than the maximum number of processes allowed to crash. However, since we will only need one fall guard for this purpose, it follows that in practice we can hope to have very few fall guards in a threshold automaton.

Since the reduction of Theorem 3 produces threshold automata with constantly many fall guards, we need another restriction on this class as well, which we now describe.

► **Definition 5.** *A threshold automaton  $\text{TA}$  over an environment  $\text{Env} = (\Pi, RC, \text{Num})$  is called multiplicative if every fall guard is simple and for every  $\mu \in \mathbb{N}_{>0}$ , (i) for every rational vector  $\mathbf{p} \in \mathbb{Q}_{\geq 0}^{|\Pi|}$ , if  $RC(\mathbf{p})$  is true then  $RC(\mu \cdot \mathbf{p})$  is true and  $\text{Num}(\mu \cdot \mathbf{p}) = \mu \cdot \text{Num}(\mathbf{p})$  and (ii) for every guard  $g := b \cdot x \bowtie a_0 + a_1 p_1 + \dots + a_l p_l$  in  $\text{TA}$  where  $\bowtie \in \{\geq, <\}$ , if  $(y, q_1, \dots, q_l)$  is a rational solution to  $g$  then  $(\mu \cdot y, \mu \cdot q_1, \dots, \mu \cdot q_l)$  is also a solution to  $g$ .*

To the best of our knowledge, many algorithms discussed in the literature (For example, see [8, 28, 6, 27, 19, 14, 7]), and more than two-thirds of all of the benchmarks of [24] satisfy multiplicativity. The main result of this section is

► **Theorem 6.** *Given a multiplicative threshold automaton  $\text{TA}$  with a constant number of fall guards and a specification  $\mathcal{L}_{\text{spec}}$ , it can be decided in time  $f(|\Phi|) \cdot n^{O(1)}$  whether there is a run of  $\text{TA}$  satisfying  $\mathcal{L}_{\text{spec}}$ .*

The rest of this section is devoted to proving this result, which we do so in four parts. Let us fix a threshold automaton  $\text{TA} = (\mathcal{L}, \mathcal{I}, \Gamma, \mathcal{R})$ , an environment  $\text{Env} = (\Pi, RC, \text{Num})$  and a specification  $\mathcal{L}_{\text{spec}}$  for the rest of this section. Let  $\Phi$  denote the set of all guards which appear in  $\text{TA}$ .

<sup>2</sup> Though the function  $g_i$  as given here gives very huge bounds, we show in the experimental section that repeated contractions can sometimes reduce the number of locations by 50%. Intuitively, this is because the number of colors of a location in the benchmarks is much smaller than the worst-case analysis performed here.

## First part: Decomposing paths into steady paths

First, similar to the paper [22], we show that the job of finding a path satisfying  $\mathcal{L}_{\text{spec}}$  can be reduced to that of finding a bounded number of concatenated “steady” paths. However, the result needs to be stated in a different manner than [22], so that later on, we could leverage the fact that the threshold automaton TA contains only constantly many fall guards.

A *context*  $\omega$  is any subset of the guards of TA, i.e.,  $\omega \subseteq \Phi$ . A rule  $r$  is said to be *activated* by a context  $\omega$  if all the rise guards of  $r$  are present in  $\omega$  and all the fall guards of  $r$  are **not** present in  $\omega$ . The set of all rules activated by a context  $\omega$  is denoted by  $\mathcal{R}_\omega$ .

The *context* of a configuration  $\sigma$ , denoted by  $\omega(\sigma)$ , is the set of all *rise* guards that evaluate to true and the set of all *fall* guards that evaluate to false in  $\sigma$ . Since the values of the shared variables can only increase along a path, it easily follows that for any configuration  $\sigma$  and any schedule  $\tau$  applicable to  $\sigma$ ,  $\omega(\sigma) \subseteq \omega(\tau(\sigma))$ .

We say that  $\text{path}(\sigma, \tau)$  is  $\omega$ -*steady* if all the rules in the schedule  $\tau$  are from  $\mathcal{R}_\omega$  and for every configuration  $\sigma' \in \text{Cfgs}(\sigma, \tau)$ , we have  $\mathcal{R}_\omega \subseteq \mathcal{R}_{\omega(\sigma')}$ . Intuitively, if  $\text{path}(\sigma, \tau)$  is  $\omega$ -steady then the path only uses rules from  $\mathcal{R}_\omega$ . We have the following lemma.

- **Lemma 7.** *The specification  $\mathcal{L}_{\text{spec}}$  can be satisfied by a path of TA iff there exists  $K \leq |\Phi|$ , configurations  $\sigma_0, \sigma'_0, \dots, \sigma_K, \sigma'_K$  and contexts  $\omega_0 \subsetneq \omega_1 \subsetneq \dots \subsetneq \omega_K$  such that*
- $\sigma_0$  is an initial configuration and  $\sigma'_K$  satisfies  $\mathcal{L}_{\text{spec}}$
  - For every  $i \leq K$ , there is a  $\omega_i$ -steady path  $\sigma_i \xrightarrow{*} \sigma'_i$
  - For every  $i < K$ , if  $\mathcal{R}_{\omega_i} \subseteq \mathcal{R}_{\omega_{i+1}}$  then there is a  $\omega_i$ -steady path  $\sigma'_i \xrightarrow{*} \sigma_{i+1}$ , otherwise  $\sigma'_i \rightarrow \sigma_{i+1}$

**Proof.** (Sketch.) Clearly if there exists such configurations and contexts then there exists a path of TA which satisfies  $\mathcal{L}_{\text{spec}}$ . To prove the other direction, suppose  $\text{path}(\sigma_0, \tau)$  is a path of TA which satisfies  $\mathcal{L}_{\text{spec}}$ . Using the fact that  $\omega(\sigma') \subseteq \omega(\tau'(\sigma'))$  for any configuration  $\sigma'$  and any schedule  $\tau'$ , we can decompose  $\text{path}(\sigma_0, \tau)$  into  $\sigma_0, \tau_0, \sigma'_0, t_0, \sigma_1, \tau_1, \sigma'_1, t_1, \dots, \sigma_K, \tau_K, \sigma'_K$  such that for every  $i$ ,  $\omega(\sigma_i) = \omega(\sigma'_i)$ ,  $\omega(\sigma'_i) \subsetneq \omega(\sigma_{i+1})$  and  $t_i$  is a rule of TA. We can then prove that the configurations  $\sigma_0, \sigma'_0, \dots, \sigma_K, \sigma'_K$  and the contexts  $\omega(\sigma_0), \dots, \omega(\sigma_K)$  satisfy the required conditions. ◀

## Second part: Establishing a connection between continuous Petri nets and steady paths

Let us fix a context  $\omega$  of the threshold automaton TA for the rest of this subsection. We say that a configuration  $\sigma$  is  $\mathcal{R}_\omega$ -applicable if  $(\sigma, \vec{g}, \sigma, \vec{p})$  satisfies every guard of every rule in  $\mathcal{R}_\omega$ .

### Continuous Petri nets

To define continuous Petri nets, we will mostly reuse the same notations from [5]. A continuous Petri net  $N$  is a tuple  $(P, T, F)$  where  $P$  is a finite set of places,  $T$  is a finite set of transitions and  $F \subseteq P \times T \cup T \times P$  is the flow relation. For a transition  $t$ , let  $\bullet t = \{p : (p, t) \in F\}$  and  $t \bullet = \{p : (t, p) \in F\}$ . A marking  $M$  of  $N$  is a function  $M : P \rightarrow \mathbb{Q}_{\geq 0}$ . Intuitively a marking  $M$  assigns  $M(p)$  many *tokens* to each place  $p \in P$ . A marking is called integral if  $M(p) \in \mathbb{N}_0$  for every place  $p$ . Given a marking  $M$  and a  $k \in \mathbb{N}_{>0}$  let  $kM$  denote the marking  $kM(p) = k \cdot M(p)$ . The transition relation between two markings  $M$  and  $M'$  is defined as follows: For  $\alpha \in (0, 1]$  and  $t \in T$ , we say that  $M \xrightarrow{\alpha t} M'$  if for every  $p \in \bullet t$ ,  $M(p) \geq \alpha$  and  $M'(p) = M(p) - \alpha$  if  $p \in \bullet t \setminus t \bullet$ ,  $M'(p) = M(p) + \alpha$  if  $p \in t \bullet \setminus \bullet t$  and  $M'(p) = M(p)$  otherwise. We say that  $M \rightarrow M'$  if  $M \xrightarrow{\alpha t} M'$  for some  $\alpha$  and  $t$ . Finally we say that  $M \xrightarrow{*} M'$  if there exists  $M_1, \dots, M_{k-1}$  such that  $M \rightarrow M_1 \rightarrow \dots \rightarrow M_{k-1} \rightarrow M'$ .

### Constructing continuous Petri nets from contexts

We now construct a *continuous Petri net*  $N_\omega$  for the context  $\omega$  as follows: For every location  $\ell$  of TA, we will have a place  $p_\ell$ . Similarly for every variable  $x \in \Gamma \cup \Pi$ , we will have a place  $p_x$ . If  $r = (\ell, \ell', \phi, \vec{u})$  is a rule in  $\mathcal{R}_\omega$ , we will have a transition  $t_r$  where  $\bullet t_r = \{p_\ell\}$  and  $t_r^\bullet = \{p_{\ell'}\} \cup \{p_x : \vec{u}[x] = 1\}$ .

We note that  $N_\omega$  tries to simulate exactly the rules of  $\mathcal{R}_\omega$ , but it does not check whether the corresponding guard of a rule is true before firing it. To ensure that a proper simulation is carried out by  $N_\omega$ , we will restrict ourselves to only runs of  $N_\omega$  over *compatible markings* which are defined as follows.

A marking  $M$  of  $N_\omega$  is called a *compatible marking* if  $\sum_{\ell \in \mathcal{L}} M(p_\ell) = \text{Num}(\{M(p_x) : x \in \Pi\})$  and if for every  $x \in \Gamma \cup \Pi$ , the assignment  $x \mapsto M(p_x)$  satisfies the resilience condition *RC* and all the guards of all the rules in  $\mathcal{R}_\omega$ . Notice that to every  $\mathcal{R}_\omega$ -applicable configuration  $\sigma$  of TA we can bijectively assign a canonical *compatible integral* marking  $\mathbb{B}(\sigma)$  of  $N_\omega$  where  $(\mathbb{B}(\sigma))(p_x) = \sigma[x]$ .

► **Proposition 8.** *The following are true:*

- Suppose  $\sigma \xrightarrow{*} \sigma'$  is an  $\omega$ -steady run of TA. Then  $\mathbb{B}(\sigma) \xrightarrow{*} \mathbb{B}(\sigma')$  in  $N_\omega$ .
- Suppose  $M$  and  $M'$  are compatible markings of  $N_\omega$  such that  $M \xrightarrow{*} M'$ . Then there exists  $\mu \in \mathbb{N}_{>0}$  such that for all  $k \in \mathbb{N}_{>0}$ ,  $\mu k M$  and  $\mu k M'$  are compatible integral markings and  $\mathbb{B}^{-1}(\mu k M) \xrightarrow{*} \mathbb{B}^{-1}(\mu k M')$  is an  $\omega$ -steady run of TA.

**Proof.** (Sketch.) The first point is obvious from the definition. For the second point, if  $M := M_0 \xrightarrow{\alpha_1 t_{r_1}} M_1 \xrightarrow{\alpha_2 t_{r_2}} M_2 \dots M_{l-1} \xrightarrow{\alpha_l t_{r_l}} M_l := M'$  is a run, then by multiplying the markings by the least common multiple of the denominators of  $\{\alpha_i\}_{i \leq l} \cup \{M_i(p_x) : i \leq l, x \in \mathcal{L} \cup \Gamma \cup \Pi\}$  (which we take to be  $\mu$ ), we can get an integral run between  $\mu k M$  and  $\mu k M'$ . Using multiplicativity of TA, we can translate this back to a run of TA. ◀

### Third part: Characterizing steady paths

It was shown in ([5], Theorems 3.6 and 3.3) that there is a logic (which the authors of [5] call *convex semi-linear Horn formulas*) characterizing reachability in continuous Petri nets, whose satisfiability can be tested **in polynomial time**. Using this result, proposition 8 and multiplicativity, we show that

► **Lemma 9.** *Given a context  $\omega$ , in polynomial time we can construct a convex semi-linear Horn formula  $\phi_\omega(\mathbf{x}, \mathbf{y})$  with  $2(|\mathcal{L}| + |\Gamma| + |\Pi|)$  free variables such that*

- If  $\sigma \xrightarrow{*} \sigma'$  is an  $\omega$ -steady path of TA then  $\phi_\omega(\sigma, \sigma')$  is true
- Suppose  $\phi_\omega(M, M')$  is true. Then there exists  $\mu \in \mathbb{N}$  such that for all  $k \in \mathbb{N}$ ,  $\mu k M, \mu k M'$  are configurations of TA such that  $\mu k M \xrightarrow{*} \mu k M'$  is an  $\omega$ -steady path in TA.

► **Lemma 10.** *Given a rule  $r$  of TA, in polynomial time we can construct a convex semi-linear Horn formula  $\phi_r(\mathbf{x}, \mathbf{y})$  with  $2(|\mathcal{L}| + |\Gamma| + |\Pi|)$  free variables such that*

- If  $\sigma$  and  $\sigma'$  are configurations of TA such that  $\sigma' = r(\sigma)$ , then  $\phi_r(\sigma, \sigma')$  is true.
- Suppose  $\phi_r(M, M')$  is true. Then there exists  $\mu \in \mathbb{N}$  such that for all  $k \in \mathbb{N}$ ,  $\mu k M, \mu k M'$  are configurations of TA such that  $\mu k M' = (\mu k \cdot r)(\mu k M)$ , i.e.,  $\mu k M'$  can be obtained by applying the rule  $r$  to  $\mu k M$ , repeatedly for  $\mu k$  many steps.

### Fourth part: Bringing it all together

► **Theorem 11.** *Given a multiplicative threshold automaton TA with constant number of fall guards and a specification  $\mathcal{L}_{\text{spec}}$ , it can be decided in time  $f(|\Phi|) \cdot n^{O(1)}$  whether there is a run of TA satisfying  $\mathcal{L}_{\text{spec}}$ .*

**Proof.** (Sketch.) One can easily show that if we have a monotonically increasing context sequence  $\omega_0 \subsetneq \omega_1 \subsetneq \dots \subsetneq \omega_K$ , the size of the set  $\{j : \mathcal{R}_{\omega_j} \not\subseteq \mathcal{R}_{\omega_{j+1}}\}$  is at most  $|\Phi^{\text{fall}}|$ . Using this observation, we proceed as follows. We iterate over all  $K \leq |\Phi|$  and over all possible monotonically increasing context sequences  $\omega_0 \subsetneq \omega_1 \subsetneq \dots \subsetneq \omega_K$  of length  $K + 1$  and all possible rule sequences  $r_1, \dots, r_c$  of length  $c = \#\{j : \mathcal{R}_{\omega_j} \not\subseteq \mathcal{R}_{\omega_{j+1}}\}$ . Note that the number of such iterations is at most  $O(|\Phi| \cdot |\Phi^{\text{fall}}| \cdot |\Phi|! \cdot 2^{|\Phi|} \cdot |\mathcal{R}|^{|\Phi^{\text{fall}}|})$ . Since  $|\Phi^{\text{fall}}|$  is assumed to be a constant, the exponential dependence only lies upon  $|\Phi|$ .

A position  $0 \leq l \leq K$  is called *bad* if  $\mathcal{R}_{\omega_l} \not\subseteq \mathcal{R}_{\omega_{l+1}}$ . Let  $j_1, \dots, j_c$  be the set of all bad positions. Using lemmas 9 and 10 we can write down the following convex semi-linear Horn formula in polynomial time:

$$\xi_0(\mathbf{x}_0, \mathbf{y}_0, \mathbf{x}_1) \wedge \xi_1(\mathbf{x}_1, \mathbf{y}_1, \mathbf{x}_2) \wedge \dots \wedge \xi_{K-1}(\mathbf{x}_{K-1}, \mathbf{y}_{K-1}, \mathbf{x}_K) \wedge \xi_K(\mathbf{x}_K, \mathbf{y}_K) \quad (1)$$

where  $\xi_K(\mathbf{x}_K, \mathbf{y}_K) = \phi_{\omega_K}(\mathbf{x}_K, \mathbf{y}_K)$  and  $\xi_i$  for  $i < K$  is defined as follows: If  $i$  is a bad position, i.e., if  $i = j_l$  for some  $1 \leq l \leq c$ , then  $\xi_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}_{i+1}) = \phi_{\omega_i}(\mathbf{x}_i, \mathbf{y}_i) \wedge \phi_{r_l}(\mathbf{y}_i, \mathbf{x}_{i+1})$ . If  $i$  not a bad position, then  $\xi_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{x}_{i+1}) = \phi_{\omega_i}(\mathbf{x}_i, \mathbf{y}_i) \wedge \phi_{\omega_i}(\mathbf{y}_i, \mathbf{x}_{i+1})$

To equation (1), we also add a constraint stating that  $\mathbf{x}_0$  is an initial configuration and  $\mathbf{y}_K$  satisfies  $\mathcal{L}_{\text{spec}}$ . By proposition 8 we can then easily show that, there is a run of TA satisfying  $\mathcal{L}_{\text{spec}}$  iff in at least one iteration, the constructed formula (1) is satisfiable. ◀

## 6 NP-hardness of multiplicative threshold automata

A natural question arises from the results of the previous section. Can we do better than fixed-parameter tractability and instead solve the reachability problem for multiplicative threshold automata in polynomial time? We remark that the proof of NP-hardness of reachability for threshold automata given in [2] does not produce multiplicative threshold automata and hence does not answer this question. Nevertheless, we show that it is unlikely for reachability of multiplicative threshold automata to be in polynomial time.

► **Theorem 12.** *Coverability (and hence reachability) for multiplicative threshold automata is NP-hard even when there are no fall guards.*

**Proof.** We give an easy reduction from 3-SAT. Let  $\varphi$  be a propositional formula with variables  $x_1, \dots, x_k$  and clauses  $C_1, \dots, C_m$ . We will have  $2k$  shared variables  $y_1, \dots, y_k, \bar{y}_1, \dots, \bar{y}_k$  and one environment variable  $n$ , denoting the number of processes. Incrementing  $y_i$  ( $\bar{y}_i$  resp.) corresponds to setting  $x_i$  to true (false resp). We will have  $2k + 1$  locations  $\ell_0, \ell'_0, \ell_1, \ell'_1, \dots, \ell'_{k-1}, \ell_k$ . Between  $\ell_i$  and  $\ell'_i$  we will have two rules which increment  $y_i$  and  $\bar{y}_i$  respectively. To ensure that all the processes increment the same variable, we have two rules from  $\ell'_i$  to  $\ell_{i+1}$  which test that  $y_i \geq n$  and  $\bar{y}_i \geq n$  respectively. Hence if one process increments  $y_i$  and another increments  $\bar{y}_i$ , then all the processes get stuck at  $\ell'_i$ .

Let  $\text{var}(x_i) = y_i$  and let  $\text{var}(\bar{x}_i) = \bar{y}_i$ . We will then have  $m$  locations  $\ell_{k+1}, \ell_{k+2}, \dots, \ell_{k+m}$  and the following rules between  $\ell_{k+i-1}$  and  $\ell_{k+i}$  for every  $1 \leq i \leq m$ : If the clause  $C_i$  is of the form  $a \vee b \vee c$  then there are three rules between  $\ell_{k+i-1}$  and  $\ell_{k+i}$ , each checking if  $\text{var}(a) \geq 1$ ,  $\text{var}(b) \geq 1$  and  $\text{var}(c) \geq 1$  respectively. Hence if either one of  $\text{var}(a)$  or  $\text{var}(b)$  or  $\text{var}(c)$  was incremented, the processes could move from  $\ell_{k+i-1}$  to  $\ell_{k+i}$ , otherwise all the

processes get stuck at  $\ell_{k+i-1}$ . Finally we set the initial location to be  $\ell_0$  and the specification to be  $\mathcal{L}_{=0} = \emptyset$  and  $\mathcal{L}_{>0} = \{\ell_{k+m}\}$ . It is then easy to see that  $\varphi$  is satisfied iff there is a run which satisfies  $\mathcal{L}_{\text{spec}}$ . ◀

## 7 Experiments

We implemented the contraction procedure for the acyclic threshold automata as presented in section 4 and then used the algorithm for multiplicative threshold automata presented in section 5. To leverage the solid engineering work that has been put into modern SMT solvers, we used the Z3 solver to solve the convex semi-linear Horn formulas as well as to choose a context (and rule) sequence. We applied our implementations to all the multiplicative protocols in the latest version of the benchmark of [24], which contains various algorithms taken from the distributed computing literature. For more information on the protocols, we refer the reader to the benchmark of [24].

■ **Table 1** The experiments were run on a machine with Intel® Core™ i5-7200U CPU with 7.7 GiB memory. The time limit was set to be 2 hours and the memory limit was set to be 7 GiB. TLE (MLE) means that the time limit (memory limit) exceeded for the particular benchmark.

Input	Case (if more than one)	Time, seconds		
		This paper	Algo from [2]	ByMC
<b>frb</b>		0.38	0.32	0.07
<b>frb</b>	hand-coded TA	0.29	0.31	0.16
<b>strb</b>		0.44	0.43	0.14
<b>strb</b>	hand-coded TA	0.32	0.30	0.10
<b>nbacg</b>		2.92	8.43	9.71
<b>aba</b>	Case 1	4.49	10.26	25.6
<b>aba</b>	Case 2	18.29	41.92	704.9
<b>cbc</b>	Case 1	3579.24	MLE	MLE
<b>cbc</b>	Case 2	183.61	2035.5	26.37
<b>cbc</b>	Case 3	MLE	MLE	MLE
<b>cbc</b>	Case 4	MLE	MLE	MLE
<b>cbc</b>	hand-coded TA	3.27	0.91	0.26
<b>cfls</b>	Case 1	13.81	13.53	37.09
<b>cfls</b>	Case 2	12.47	16.14	186.5
<b>cfls</b>	Case 3	84.95	86.98	7875
<b>cfls</b>	hand-coded TA	1.75	1.31	2737.53
<b>c1cs</b>	Case 1	179.39	598.2	TLE
<b>c1cs</b>	Case 2	70.77	747.86	7119.71
<b>c1cs</b>	Case 3	604.91	1575.21	MLE
<b>c1cs</b>	hand-coded TA	4.87	6.63	TLE

*Evaluation:* Table 1 summarizes our results and compares them with the results obtained using ByMC, the tool presented in [24] and the algorithm from [2].

For some safety specifications, our contraction procedure was able to reduce the number of locations by more than 50% for the cbc protocol(s). This helped us save some memory, as we also noticed that running just the algorithm for multiplicative threshold automata

took much more memory and the algorithm was not able to complete its execution. Our implementation compares favorably with both ByMC and the algorithm from [2] in some cases, but also performs worse in some of the hand-coded examples, the second case of cbc and the frb and strb protocols.

## 8 Conclusion

In this paper, we have investigated the parameterized complexity of safety in threshold automata. Though we have proved hardness results even in very restricted settings, we have also identified tractable special cases which arise in practice. A preliminary implementation of our algorithms suggest that these methods might be useful in practice as well.

For the sake of simplicity, we have only restricted to verifying safety properties in this paper. A special type of logic called  $\text{ELTL}_{\text{FT}}$  [12] has been proposed for threshold automata which can express various safety and liveness properties. Since model checking this logic decomposes to a finite number of safety specifications (modulo some technical constraints), we believe that our algorithm for multiplicative threshold automata can be adapted to give an algorithm for model checking this logic as well.

---

## References

- 1 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006. doi:10.1007/s00446-005-0138-3.
- 2 A. R. Balasubramanian, Javier Esparza, and Marijana Lazić. Complexity of verification and synthesis of threshold automata. In *Accepted at ATVA 2020*, 2020. URL: <https://arxiv.org/abs/2007.06248>.
- 3 Nathalie Bertrand, Igor Konnov, Marijana Lazić, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. In *CONCUR*, volume 140 of *LIPICs*, pages 33:1–33:15, 2019.
- 4 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- 5 Michael Blondin and Christoph Haase. Logics for continuous reachability in petri nets and vector addition systems with states. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005068.
- 6 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.
- 7 Francisco Vilar Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *PaCT*, volume 2127 of *LNCS*, pages 42–50, 2001.
- 8 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
- 9 Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. On the complexity of bounded context switching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPICs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.27.
- 10 Peter Chini, Roland Meyer, and Prakash Saivasan. Fine-grained complexity of safety verification. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki*,

- Greece, April 14-20, 2018, *Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2018. doi:10.1007/978-3-319-89963-3\_2.
- 11 Peter Chini, Roland Meyer, and Prakash Saivasan. Complexity of liveness in parameterized systems. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 37:1–37:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.37.
  - 12 Peter Chini, Roland Meyer, and Prakash Saivasan. Liveness in broadcast networks. In *NETYS 2019, Revised Selected Papers*, pages 52–66, 2019.
  - 13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
  - 14 Dan Dobre and Neeraj Suri. One-step consensus with zero-degradation. In *DSN*, pages 137–146, 2006.
  - 15 Constantin Enea and Azadeh Farzan. On atomicity in presence of non-atomic writes. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 497–514. Springer, 2016. doi:10.1007/978-3-662-49674-9\_29.
  - 16 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS*, pages 352–359. IEEE Computer Society, 1999.
  - 17 Azadeh Farzan and P. Madhusudan. The complexity of predicting atomicity violations. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2009. doi:10.1007/978-3-642-00768-2\_14.
  - 18 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.
  - 19 Rachid Guerraoui. Non-blocking atomic commit in asynchronous distributed systems with failure detectors. *Distributed Computing*, 15(1):17–25, 2002.
  - 20 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.
  - 21 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704 of *LNCS*, pages 125–140, 2014.
  - 22 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015. doi:10.1007/978-3-319-21690-4\_6.
  - 23 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Information and Computation*, 252:95–109, 2017.
  - 24 Igor Konnov and Josef Widder. Bymc: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.
  - 25 Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL 2017*, pages 719–734, 2017.

- 26 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In *CONCUR*, pages 19:1–19:17, 2018.
- 27 Achour Mostéfaoui, Eric Mourgaya, Philippe Raipin Parvédy, and Michel Raynal. Evaluating the condition-based approach to solve consensus. In *DSN*, pages 541–550, 2003.
- 28 T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Dist. Comp.*, 2:80–94, 1987.



# Uncertainty Reasoning for Probabilistic Petri Nets via Bayesian Networks

**Rebecca Bernemann**

University of Duisburg-Essen, Germany  
rebecca.bernemann@uni-due.de

**Benjamin Cabrera**

University of Duisburg-Essen, Germany  
benjamin.cabrera@uni-due.de

**Reiko Heckel**

University of Leicester, UK  
rh122@leicester.ac.uk

**Barbara König**

University of Duisburg-Essen, Germany  
barbara\_koenig@uni-due.de

---

## Abstract

This paper exploits extended Bayesian networks for uncertainty reasoning on Petri nets, where firing of transitions is probabilistic. In particular, Bayesian networks are used as symbolic representations of probability distributions, modelling the observer’s knowledge about the tokens in the net. The observer can study the net by monitoring successful and failed steps.

An update mechanism for Bayesian nets is enabled by relaxing some of their restrictions, leading to modular Bayesian nets that can conveniently be represented and modified. As for every symbolic representation, the question is how to derive information – in this case marginal probability distributions – from a modular Bayesian net. We show how to do this by generalizing the known method of variable elimination. The approach is illustrated by examples about the spreading of diseases (SIR model) and information diffusion in social networks. We have implemented our approach and provide runtime results.

**2012 ACM Subject Classification** Mathematics of computing → Bayesian networks; Software and its engineering → Petri nets

**Keywords and phrases** uncertainty reasoning, probabilistic knowledge, Petri nets, Bayesian networks

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.38

**Related Version** A full version of the paper is available as [1], <https://arxiv.org/abs/2009.14817>.

**Funding** This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant GRK 2167, Research Training Group “User-Centred Social Media”.

## 1 Introduction

Today’s software systems and the real-world processes they support are often distributed, with agents acting independently based on their own local state but without complete knowledge of the global state. E.g., a social network may expose a partial history of its users’ interactions while hiding their internal states. An application tracing the spread of a virus can record test results but not the true infection state of its subjects. Still, in both cases, we would like to derive knowledge under uncertainty to allow us, for example, to predict the spread of news in the social network or trace the outbreak of a virus.



© Rebecca Bernemann, Benjamin Cabrera, Reiko Heckel, and Barbara König;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 38; pp. 38:1–38:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Using Petri nets as a basis for modelling concurrent systems, our aim is to perform uncertainty reasoning on Petri nets, employing Bayesian networks as compact representations of probability distributions. Assume that we are observing a discrete-time concurrent system modelled by a Petri net. The net’s structure is known, but its initial state is uncertain, given only as an a-priori probability distribution on markings. The net is probabilistic: Transitions are chosen at random, either from the set of enabled transitions or independently, based on probabilities that are known but may change between steps. We cannot observe which transition actually fires, but only if firing was successful or failed. Failures occur if the chosen transition is not enabled under the current marking (in the case where we choose transitions independently), if no transition can fire, or if a special fail transition is chosen. After observing the system for a number of steps, recording a sequence of “success” and “failure” events, we then determine a marginal distribution on the markings (e.g., compute the probability that a given place is marked), taking into account all observations.

First, we set up a framework for uncertainty reasoning based on time-inhomogeneous Markov chains that formally describes this scenario, parameterized over the specific semantics of the probabilistic net. This encompasses the well-known stochastic Petri nets [29], as well as a semantics where the choice of the marking and the transition is independent (Sct. 2 and 3). Using basic Bayesian reasoning (reminiscent of methods used for hidden Markov models [32]), it is conceptually relatively straightforward to update the probability distribution based on the acquired knowledge. However, the probability space is exponential in the number of places of the net and hence direct computations become infeasible relatively quickly.

Following [5], our solution is to use (modular) Bayesian networks [36, 13, 31] as compact symbolic representations of probability distributions. Updates to the probability distribution can be performed very efficiently on this data structure, simply by adding additional nodes. By analyzing the structure of the Petri net we ensure that this node has a minimal number of connections to already existing nodes (Sct. 4 and 5).

As for every symbolic representation, the question is how to derive information, in this case marginal probability distributions. We solve this question by generalizing the known method of variable elimination [14, 13] to modular Bayesian networks. This method is known to work efficiently for networks of small treewidth, a fact that we experimentally verify in our implementation (Sct. 6 and 7).

We consider some small application examples modelling gossip and infection spreading. Summarized, our contributions are:

- We propose a framework for uncertainty reasoning based on time-inhomogeneous Markov chains, parameterized over different types of probabilistic Petri nets (Sct. 2 and 3).
- We use modular Bayesian networks to symbolically represent and update probability distributions (Sct. 4 and 5).
- We extend the variable elimination method to modular Bayesian networks and show how it can be efficiently employed in order to compute marginal distributions (Sct. 6). This is corroborated by our implementation and runtime results (Sct. 7).

All proofs and further material can be found in the full version [1].

## **2 Markov Chains and Probabilistic Condition/Event Nets**

### **2.1 Markov Chains**

Markov chains [18, 35] are a stochastic state-based model, in which the probability of a transition depends only on the state of origin. Here we restrict to a finite state space.

► **Definition 1** (Markov chain). *Let  $Q$  be a finite state space. A (discrete-time) Markov chain is a sequence  $(X_n)_{n \in \mathbb{N}_0}$  of random variables such that for  $q, q_0, \dots, q_n \in Q$ :*

$$P(X_{n+1} = q \mid X_n = q_n) = P(X_{n+1} = q \mid X_n = q_n, \dots, X_0 = q_0).$$

Assume that  $|Q| = k$ . Then, the probability distribution over  $Q$  at time  $n$  can be represented as a  $k$ -dimensional vector  $p^n$ , indexed over  $Q$ . We abbreviate  $p^n(q) = P(X_n = q)$ . We define  $k \times k$ -transition matrices  $P^n$ , indexed over  $Q$ , with entries<sup>1</sup> for the entry of matrix  $M$  at row  $q'$  and column  $q$ :  $P^n(q' \mid q) = P(X_{n+1} = q' \mid X_n = q)$ . Note that  $p^{n+1} = P^n \cdot p^n$ . We do not restrict to time-homogeneous Markov chains where it is required that  $P^n = P^{n+1}$  for all  $n \in \mathbb{N}_0$ . Instead, the probability distribution on the transitions might vary over time.

## 2.2 Probabilistic Condition/Event Nets

As a basis for probabilistic Petri nets we use the following variant of condition/event nets [33]. Deviating from [33], we omit the initial marking and furthermore the fact that the post-condition is marked is not inhibiting the firing of a transition. That is, we omit the so-called contact condition, which makes it easier to model examples from application scenarios where the contact condition would be unnatural. Note however that we could easily accommodate the theory to include this condition, as we did in the predecessor paper [5].

► **Definition 2** (condition/event net). *A condition/event net (C/E net or simply Petri net)  $N = (S, T, \bullet(\cdot), (\cdot)^\bullet)$  is a four-tuple consisting of a finite set of places  $S$ , a finite set of transitions  $T$  with pre-conditions  $\bullet(\cdot) : T \rightarrow \mathcal{P}(S)$  and post-conditions  $(\cdot)^\bullet : T \rightarrow \mathcal{P}(S)$ . A marking is any subset of places  $m \subseteq S$  and will also be represented by a bit string  $m \in \{0, 1\}^{|S|}$  (assuming an ordering on the places).*

*A transition  $t$  can fire for a marking  $m \subseteq S$  if  $\bullet t \subseteq m$ . Then marking  $m$  is transformed into  $m' = (m \setminus \bullet t) \cup t^\bullet$ , written  $m \xrightarrow{t} m'$ . We write  $m \xrightarrow{t}$  to indicate that there exists some  $m'$  with  $m \xrightarrow{t} m'$  and  $m \not\xrightarrow{t}$  if this is not the case. We denote the set of all markings by  $\mathcal{M} = \mathcal{P}(S)$ .*

In order to obtain a Markov chain from a C/E net, we need the following data: given a marking  $m$  and a transition  $t$ , we denote by  $r_n(m, t)$  the probability of firing  $t$  in marking  $m$  (at step  $n$ ), and by  $r_n(m, \text{fail})$  the probability of going directly to a fail state  $*$ .

► **Definition 3.** *Let  $N = (S, T, \bullet(\cdot), (\cdot)^\bullet)$  be a condition/event net and let  $T_f = T \cup \{\text{fail}\}$  (the set of transitions enriched with a fail transition). Furthermore let  $r_n : \mathcal{M} \times T_f \rightarrow [0, 1]$ ,  $n \in \mathbb{N}_0$  be a family of functions (the transition distributions at step  $n$ ), such that for each  $n \in \mathbb{N}_0$ ,  $m \in \mathcal{M}$ :  $\sum_{t \in T_f} r_n(m, t) = 1$ .*

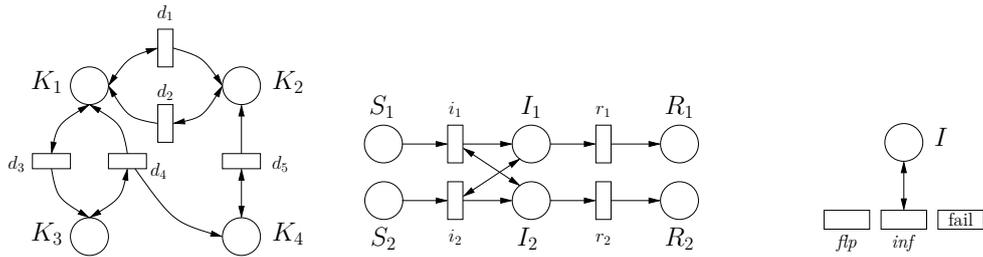
*The Markov chain generated from  $N, r_n$  has states  $Q = \mathcal{M} \cup \{*\}$  and for  $m, m' \in \mathcal{M}$ :*

$$\begin{aligned} P(X_{n+1} = m' \mid X_n = m) &= \sum_{t \in T, m \xrightarrow{t} m'} r_n(m, t) & P(X_{n+1} = m' \mid X_n = *) &= 0 \\ P(X_{n+1} = * \mid X_n = m) &= \sum_{t \in T_f, m \not\xrightarrow{t}} r_n(m, t) & P(X_{n+1} = * \mid X_n = *) &= 1 \end{aligned}$$

where we assume that  $m \not\xrightarrow{\text{fail}}$  for every  $m \in \mathcal{M}$ .

Note that we can make a transition from  $m$  to the fail state  $*$  either when there is a non-zero probability for performing such a transition directly or when we pick a transition that

<sup>1</sup> We are using the notation  $M(q' \mid q)$ , resembling conditional probability,



(a) A Petri net modelling gossip diffusion in a social network ( $K_i$ :  $i$  knows information).

(b) A Petri net modelling spread of a disease ( $S$ : susceptible,  $I$ : infected,  $R$ : removed).

(c) A Petri net modelling a test with false positives and negatives ( $I$ : infected).

■ **Figure 1** Example Petri nets.

cannot be fired in  $m$ . Requiring that  $m \not\Rightarrow^{\text{fail}}$  for every  $m$  is for notational convenience, since we have to sum up all probabilities leading to the fail state  $*$  to compute  $P(X_{n+1} = * | X_n = m)$ . In this way the symbol  $\not\Rightarrow$  always signifies a transition to  $*$ .

By parametrising over  $r_n$  we obtain different semantics for condition/even nets. In particular, we consider the following two probabilistic semantics, both based on probability distributions  $p_T^n: T \rightarrow [0, 1]$ ,  $n \in \mathbb{N}_0$  on transitions. We work under the assumption that this information is given or can be gained from extra knowledge that we have about our environment.

**Independent case:** Here we assume that the marking and the transition are drawn independently, where markings are distributed according to  $p^n$  and transitions according to  $p_T^n$ . It may happen that the transition and the marking do not “match” and the transition cannot fire. Formally,  $r_n(m, t) = p_T^n(t)$ ,  $r_n(m, \text{fail}) = 0$  (where  $m \in \mathcal{M}$ ,  $t \in T$ ). This extends to the case where fail has non-zero probability, with probability distribution  $p_T^n: T_f \rightarrow [0, 1]$ .

**Stochastic net case:** We consider stochastic Petri nets [29] which are often provided with a semantics based on continuous-time Markov chains [35]. Here, however we do not consider continuous time, but instead model the embedded discrete-time Markov chain of jumps that abstracts from the timing. The firing rate of a transition  $t$  is proportional to  $p_T^n(t)$ .

Intuitively, we first sample a marking  $m$  (according to  $p^n$ ) and then sample a transition, restricting to those that are enabled in  $m$ . Formally, for every  $t \in T_f$ ,  $r_n(m, t) = 0$ ,  $r_n(m, \text{fail}) = 1$  if no transition can fire in  $m$  and  $r_n(m, t) = p_T^n(t) / \sum_{m \Rightarrow t'} p_T^n(t')$ ,  $r_n(m, \text{fail}) = 0$  otherwise.

Other semantics might make sense, for instance the probability of firing a transition could depend on a place not contained in its pre-condition. Furthermore, it is possible to mix the two semantics and do one step in the independent and the next in the stochastic semantics.

► **Example 1.** The following nets illustrate the two semantics. The first net (Fig. 1a) explains the diffusion of gossip in a social network: There are four users and each place  $K_i$  represents the knowledge of user  $i$ . To convey the fact that user  $i$  knows some secret, place  $K_i$  contains a token. The diffusion of information is represented by transitions  $d_j$ . E.g., if 1 knows the secret he will tell it to either 2 or 3 and if 3 knows a secret she will broadcast it to both 1 and 4. Note that a person will share the secret even if the recipient already knows, and she will retain this knowledge (see the double arrows in the net).<sup>2</sup>

<sup>2</sup> Hence, in the Petri net semantics, we allow a transition to fire although the post-conditions is marked.

Here we use the stochastic semantics: only transitions that are enabled will be chosen (unless the marking is empty and no transition can fire). We assume that  $p_T(d_2) = 1/3$  and  $p_T(d_1) = p_T(d_3) = p_T(d_4) = p_T(d_5) = 1/6$ , i.e., user 2 is more talkative than the others.

One of the states of the Markov chain is the marking  $m = 1100$  ( $K_1, K_2$  are marked – users 1 and 2 know the secret – and  $K_3, K_4$  are unmarked – users 3 and 4 do not). In this situation transitions  $d_1, d_2, d_3$  are enabled. We normalize the probabilities and obtain that  $d_2$  fires with probability  $1/2$  and the other two with probability  $1/4$ . By firing  $d_1$  or  $d_2$  we stay in state 1100, i.e., the corresponding Markov chain has a loop with probability  $3/4$ . Firing  $d_3$  gives us a transition to state 1110 (user 3 now knows the secret too) with probability  $1/4$ .

The second net (Fig. 1b) models the classical SIR infection model [24] for two persons. A person is *susceptible* (represented by a token in place  $S_i$ ) if he or she has not yet been infected. If the other person is infected (i.e. place  $I_1$  or  $I_2$  is marked), then he or she might also get *infected* with the disease. Finally, people recover (or die), which means that they are *removed* (places  $R_i$ ). Again we use the stochastic semantics.

The third net (Fig. 1c) models a test (for instance for an infection) that may have false positives and false negatives. A token in place  $I$  means that the corresponding person is infected. Apart from  $I$  there is another random variable  $R$  (for result) that tells whether the test is positive or negative. In order to faithfully model the test, we assign the following probabilities to the transitions:  $p_T(flp) = P(R | \bar{I})$  (false or lucky positive: this transition can fire regardless of whether  $I$  is marked, in which case the test went wrong and is only accidentally positive),  $p_T(inf) = P(R | I) - P(R | \bar{I})$  (the remaining probability,<sup>3</sup> such that the probabilities of *flp* and *inf* add up to the true positive) and  $p_T(fail) = P(\bar{R} | I)$  (false negative). Here we use the independent semantics, assuming that we have a random test where the ground truth (infected or not infected) is independent of the firing probabilities of the transitions.

### 3 Uncertainty Reasoning for Condition/Event Nets

We now introduce the following scenario for uncertainty reasoning: assume that we are given an initial probability distribution  $p_*^0$  on the markings of the Petri net. We stipulate that the fail state  $*$  cannot occur, assuming that the state of the net is always some (potentially unknown) well-defined marking. If this fail state would be reached in the Markov model, we assume that the marking of the Petri net does not change, i.e., we perform a “reset” to the previous marking.

Furthermore, we are aware of all firing probabilities of the various transitions, given by the functions  $(r_n)_{n \in \mathbb{N}_0}$  and hence all transition matrices  $P^n$  that specify the transition probabilities at step  $n$ .

Then we observe the system and obtain a sequence of *success* and *failure* occurrences. We are not told which exact transition fires, but only if the firing is successful or fails (since the pre-condition of the transition is not covered by the marking). Note that according to our model, transitions *can* be chosen to fire, although they are not activated. This could happen if either a user or the environment tries to fire such a transition, unaware of the status of its pre-condition. Failure corresponds to entering state  $*$  and in this case we assume the marking does not change. That is, we keep the previous marking, but acquire additional knowledge – namely that firing fails – which is used to update the probability distribution according to Prop. 4 (by performing the corresponding matrix multiplications, including normalization).

<sup>3</sup> Here we require that  $P(R | \bar{I}) \leq P(R | I)$ .

We use the following notation: let  $M$  be a matrix indexed over  $\mathcal{M} \cup \{*\}$ . Then we denote by  $M_*$  the matrix obtained by deleting the  $*$ -indexed row and column from  $M$ . Analogously for a vector  $p$ . Note that  $(M \cdot p)_* = M_* \cdot p_*$ . Furthermore if  $p_*$  is a sub-probability vector, indexed over  $\mathcal{M}$ ,  $\text{norm}(p_*)$  stands for the corresponding normalized vector, where the  $m$ -entry is  $p_*(m) / (\sum_{m' \in \mathcal{M}} p_*(m'))$ .

► **Proposition 4.** Let  $r_n: \mathcal{M} \times T_f \rightarrow [0, 1]$  and  $p^n: \mathcal{M} \cup \{*\} \rightarrow [0, 1]$  be given as above. Let  $N$  be a C/E net and let  $(X_n)_{n \in \mathbb{N}_0}$  be the Markov chain generated from  $N, r_n$ . Then

- $P(X_{n+1} = m' \mid X_{n+1} \neq *, X_n \neq *) = P(X_{n+1} = m' \mid X_{n+1} \neq *) = \text{norm}(P_*^n \cdot p_*^n)(m')$
- $P(X_n = m \mid X_{n+1} = *, X_n \neq *) = \text{norm}(F_*^n \cdot p_*^n)(m)$

where  $p^n(m) = P(X_n = m)$ ,  $p^n(*) = P(X_n = *)$  and  $F^n$  is a diagonal matrix with  $F^n(\bar{m} \mid \bar{m}) := P^n(* \mid \bar{m})$ ,  $\bar{m} \in \mathcal{M}$ , and  $F^n(* \mid *) := P^n(* \mid *) = 1$ , all other entries are 0.

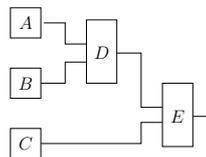
Hence, in case we observe a success we update the probability distribution to  $\bar{p}_{n+1}$  by computing  $P_*^n \cdot \bar{p}^n$  (and normalizing). Instead, in the case of a failure we assume that the marking stays unchanged, but by observing the failure we have gathered additional knowledge, which means that we can replace  $\bar{p}_{n+1}$  by  $F_*^n \cdot \bar{p}^n$  (after normalization).

$P_*^n$  and  $F_*^n$  are typically not stochastic, but only sub-stochastic. For a (sub-)probability matrix  $M_*$  and a (sub-)probability vector  $p_*$  it is easy to see that  $\text{norm}(M_* \cdot p_*) = \text{norm}(M_* \cdot \text{norm}(p_*))$ . Hence another option is to omit the normalization steps and to normalize at the very end of the sequence of observations. Normalization may be undefined (in the case of the 0-vector), which signifies that we assumed an a priori probability distribution that is inconsistent with reality.

► **Example 2.** We get back to Ex. 1 and discuss uncertainty reasoning. Assume that in the net in Fig. 1b person 1 is susceptible ( $S_1$  is marked), person 2 is infected ( $I_2$  is marked) and the  $i_j$ -transitions have a higher rate (higher probability of firing) than the  $r_j$ -transitions. Then, in the next step the probability that both are infected is higher than the probability that 1 is still susceptible and 2 has recovered.

Regarding the net in Fig. 1c we can show that in the next step, in the case of success, the probability distribution is updated in such a way that place  $I$  is marked with probability  $P(I \mid R)$  and unmarked with probability  $P(\bar{I} \mid R)$  ( $P(I \mid \bar{R})$ ,  $P(\bar{I} \mid \bar{R})$  in the case of failure), exactly as required. For more details see [1].

## 4 Modular Bayesian Networks



■ **Figure 2** An example Bayesian network.

In order to implement the updates to the probability distributions described above in an efficient way, we will now represent probability distributions over markings symbolically as Bayesian networks [31, 8]. Bayesian networks (BNs) model certain probabilistic dependencies of random variables through conditional probability tables and a graphical representation.

Consider for instance the Bayesian network in Fig. 2. Each node ( $A, B, C, D, E$ ) represents a binary random variable, where a node without predecessors (e.g.,  $A$ ) is associated with the probabilities  $P(A)$  and  $P(\bar{A})$ . Edges denote dependencies: for instance  $D$  is

dependent on  $A, B$ , which means that  $D$  is associated with a conditional probability table (matrix) with entries  $P(D | A, B)$ , similar for  $E$  (entries of the form  $P(E | D, C)$ ). In both cases, the matrix contains  $2 \cdot 4 = 8$  entries.

We will later describe how to derive probability distributions and marginal probabilities (for instance  $P(E)$ ) from a Bayesian network.

We deviate from the literature on Bayesian networks in three respects: first, since we will update and transform those networks, we need a structure where we can easily express compositionality via sequential and parallel composition. To this end we use the representation of Bayesian networks via PROPs as in [16, 22]. Second, we permit sub-stochastic matrices. Third, we allow a node to have several outgoing wires, whereas in classical Bayesian networks a node is always associated to the distribution of a single random variable. This is needed since we need to add nodes to a network that represent stochastic matrices of arbitrary dimensions (basically the matrices  $P^n$  and  $F^n$  of Proposition 4). We rely on the notation introduced in [5], but extend it by taking the last item above into account.

## 4.1 Causality Graphs

The syntax of Bayesian networks is provided by *causality graphs* [5]. For this we fix a set of node labels  $G$ , also called *generators*, where every  $g \in G$  is associated with a type  $n_g \rightarrow m_g$ , where  $n_g, m_g \in \mathbb{N}_0$ .

► **Definition 5 (Causality Graph (CG)).** A causality graph (CG) of type  $n \rightarrow m$ ,  $n, m \in \mathbb{N}_0$ , is a tuple  $B = (V, \ell, s, \text{out})$  where

- $V$  is a set of nodes
  - $\ell: V \rightarrow G$  is a labelling function that assigns a generator  $\ell(v) \in G$  to each node  $v \in V$ .
  - $s: V \rightarrow W_B^*$  is the source function that maps a node to a sequence of input wires, where  $|s(v)| = n_{\ell(v)}$  and  $W_B = \{(v, p) \mid v \in V, p \in \{1, \dots, m_{\ell(v)}\}\} \cup \{i_1, \dots, i_n\}$  is the wire set.
  - $\text{out}: \{o_1, \dots, o_m\} \rightarrow W_B$  is the output function that assigns each output port to a wire.
- Moreover, the corresponding directed graph (defined by  $s$ ) has to be acyclic.

We also define the target function  $t: V \rightarrow W_B^*$  with  $t(v) = (v, 1) \dots (v, m_{\ell(v)})$  and the set of internal wires  $IW_B = W_B \setminus \{i_1, \dots, i_n, \text{out}(o_1), \dots, \text{out}(o_m)\}$ .

We visualize such causality graphs by drawing the  $n$  input wires on the left and the  $m$  outputs on the right. Each node  $v$  is drawn as a box, with  $n_v$  ingoing wires and  $m_v$  outgoing wires, ordered from top to bottom. Connections induced by the source and by the output function are drawn as undirected edges (see Fig. 2).

We define two operations on causality graphs: sequential composition and tensor. Given  $B$  of type  $n \rightarrow k$  and  $B'$  of type  $k \rightarrow m$ , the sequential composition is obtained via concatenation, by identifying the output wires of  $B$  with the input wires of  $B'$ , resulting in  $B; B'$  of type  $n \rightarrow m$ . The tensor takes two causality graphs  $B_i$  of type  $n_i \rightarrow m_i$ ,  $i \in \{1, 2\}$  and takes their disjoint union, concatenating the sequences of input and output wires, resulting in  $B_1 \otimes B_2$  of type  $n_1 + n_2 \rightarrow m_1 + m_2$ . For formal definitions see [5, 4].

## 4.2 (Sub-)Stochastic Matrices

The semantics of modular Bayesian networks is given by (*sub-*)*stochastic matrices*, i.e., matrices with entries from  $[0, 1]$ , where column sums will be at most 1. If the sum equals exactly 1 we obtain stochastic matrices.

We consider only matrices whose dimensions are a power of two. Analogously to causality graphs, we type matrices, and say that a matrix has type  $n \rightarrow m$  whenever it is of dimension  $2^m \times 2^n$ . We again use a sequential composition operator ; that corresponds to *matrix*

multiplication ( $P; Q = Q \cdot P$ ) and the Kronecker product  $\otimes$  as the tensor. More concretely, given  $P: n_1 \rightarrow m_1$ ,  $Q: n_2 \rightarrow m_2$  we define  $P \otimes Q: n_1 + n_2 \rightarrow m_1 + m_2$  as  $(P \otimes Q)(\mathbf{x}_1 \mathbf{x}_2 | \mathbf{y}_1 \mathbf{y}_2) = P(\mathbf{x}_1 | \mathbf{y}_1) \cdot Q(\mathbf{x}_2 | \mathbf{y}_2)$  where  $\mathbf{x}_i \in \{0, 1\}^{m_i}$ ,  $\mathbf{y}_i \in \{0, 1\}^{n_i}$ .

### 4.3 Modular Bayesian Networks

Finally, *modular Bayesian networks*, adapted from [5], are causality graphs, where each generator  $g \in G$  is associated with a (sub-)stochastic matrix of suitable type.

► **Definition 6** (Modular Bayesian network (MBN)). *An MBN is a tuple  $(B, ev)$  where  $B$  is a causality graph and  $ev$  an evaluation function that assigns to every generator  $g \in G$  of type  $n \rightarrow m$  a  $2^m \times 2^n$  sub-stochastic matrix  $ev(g)$ . An MBN  $(B, ev)$  is called an ordinary Bayesian network (OBN) whenever  $B$  has no inputs (i.e. it has type  $0 \rightarrow m$ ), each generator is of type  $n \rightarrow 1$ , out is a bijection and every node is associated with a stochastic matrix.*

We now describe how to evaluate an MBN to obtain a (sub-)stochastic matrix. For OBNs – which are exactly the Bayesian networks considered in [17] – this coincides with the standard interpretation and yields a probability vector of dimension  $m$ .

► **Definition 7** (MBN evaluation). *Let  $(B, ev)$  be an MBN where  $B$  is of type  $n \rightarrow m$ . Then  $M_{ev}(B)$  is a  $2^m \times 2^n$ -matrix, which is defined as follows:*

$$M_{ev}(B)(x_1 \dots x_m | y_1 \dots y_n) = \sum_{b \in \mathcal{B}} \prod_{v \in V} ev(l(v))(b(t(v)) | b(s(v)))$$

with  $x_1, \dots, x_m, y_1, \dots, y_n \in \{0, 1\}$ .  $\mathcal{B}$  is the set of all functions  $b: W_B \rightarrow \{0, 1\}$  such that  $b(i_j) = y_j$ ,  $b(out(o_k)) = x_k$ , where  $k \in \{1, \dots, m\}$ ,  $j \in \{1, \dots, n\}$ . The functions  $b$  are applied pointwise to sequences of wires.

Calculating the underlying probability distribution of an MBN can also be done on a graphical level by treating every occurring wire as a boolean variable that can be assigned either 0 or 1. Function  $b \in \mathcal{B}$  assigns the wires, ensuring consistency with the input/output values. After the wire assignment, the corresponding entries of each matrix  $ev(l(v))$  are multiplied. After iterating over every possible wire assignment, the products are summed up.

Note that  $M_{ev}$  is compositional, it preserves sequential composition and tensor. More formally, it is a functor between symmetric monoidal categories, or – more specifically – between CC-structured PROPs (More details on PROPs are given in the full version [1]).

► **Example 3.** We illustrate Def. 7 by evaluating the Bayesian network  $(B', ev)$  in Fig. 2. This results in a  $2 \times 1$ -matrix  $M_{ev}(B')$ , assigning (sub-)probabilities to the only output wire in the diagram being 1 or 0, respectively. More concretely, we assign values to the four inner wires to obtain:

$$M_{ev}(B')(e) = \sum_{a \in \{0,1\}} \sum_{b \in \{0,1\}} \sum_{c \in \{0,1\}} \sum_{d \in \{0,1\}} (A(a) \cdot B(b) \cdot C(c) \cdot D(d | ab) \cdot E(e | cd)),$$

where  $a, b, c, d, e$  correspond to the output wire of the corresponding matrix  $(A, B, C, D, E)$ .

## 5 Updating Bayesian Networks

An MBN  $B$  of type  $0 \rightarrow k$ , as defined above, symbolically represents a probability distribution on  $\{0, 1\}^k$ , that is, a probability distribution on markings of a net with  $|S| = k$  places.

Under uncertainty reasoning (cf. Section 3), the probability distribution in the next step  $p^{n+1}$  is obtained by multiplying  $p^n$  with a matrix  $M$  (either  $P_*^n$  in the successful case or  $F_*^n$  in the case of failure). Hence, a simple way to update  $B$  would be to create an MBN  $B_M$  with a single node  $v$  (labelled by a generator  $g$  with  $ev(g) = M$ ), connected to  $k$  inputs and  $k$  outputs. Then the updated  $B'$  is simply  $B; B_M$  (remember that sequential composition corresponds to matrix multiplication). However, at dimension  $2^k \times 2^k$  the matrix  $M$  is huge and we would sacrifice the desirable compact symbolic representation. Hence the aim is to decompose  $M = M' \otimes \text{Id}$  where  $\text{Id}$  is an identity matrix of suitable dimension. Due to the functoriality of MBN evaluation this means composing with a smaller matrix and a number of identity wires (see e.g. Fig. 3b).

This decomposition arises naturally from the structure of the Petri net  $N$ , in particular if there are only relatively few transitions that may fire in a step. In this case we intuitively have to attach a stochastic matrix only to the wires representing the places connected to those transitions, while the other wires can be left unchanged. If there are several updates, we of course have to attach several matrices, but each of them might be of a relatively modest size.

In order to have a uniform treatment of the various semantics, we assume that for each step  $n$  there is a set  $\bar{S} \subseteq S$  of places<sup>4</sup> and a set  $\bar{T} \subseteq T_f$  of transitions such that: (i)  $r_n(m, t) = 0$  whenever  $t \notin \bar{T}$ ; (ii)  $r_n(m_1 m_2, t) = \bar{r}(m_1, t)$  for some function  $\bar{r}$  (where  $m_1$  is a marking of length  $\ell = |\bar{S}|$ , corresponding to the places of  $\bar{S}$ ); (iii)  $\bar{S}$  contains at least  $\bullet t, t \bullet$  for all  $t \in \bar{T}$ . Intuitively,  $\bar{S}, \bar{T}$  specify the relevant places and transitions.

For the two Petri net semantics studied earlier, these conditions are satisfied if we take as  $\bar{T}$  the support of  $p_T^n$  and as  $\bar{S}$  the union of all pre- and post-sets of  $\bar{T}$ . The function  $r_n$  can in both cases be defined in terms of  $\bar{r}$ : in the independent case this is obvious, whereas in the stochastic net case we observe that  $r_n(m, t)$  is only dependent on  $p_T^n$  and on the set of transitions that is enabled in  $m$  and this can be derived from  $m_1$ .

Now, under these assumptions, we can prove that we obtain the decomposition mentioned above.

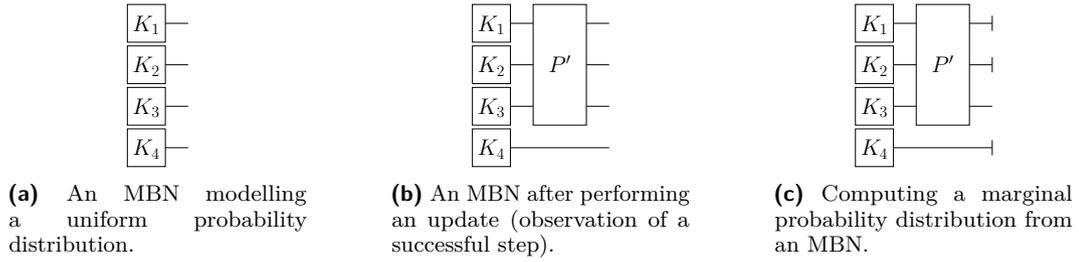
► **Proposition 8.** *Assume that  $N$  is a condition/even-net together with a function  $r_n$ . Assume that we have  $\bar{S} \subseteq S, \bar{T} \subseteq T_f$  satisfying the conditions above. Then*

- $P_*^n = P' \otimes \text{Id}_{2^{k-\ell}}$  where  $P'(m'_1 | m_1) = \sum_{t \in \bar{T}, m_1 \xrightarrow{t} m'_1} \bar{r}(m_1, t)$ .
  - $F_*^n = F' \otimes \text{Id}_{2^{k-\ell}}$  where  $F'(m'_1 | m_1) = \sum_{t \in \bar{T}, m_1 \xrightarrow{t} m'_1} \bar{r}(m_1, t)$  if  $m_1 = m'_1$  and 0 otherwise.
- Here  $P', F'$  are  $2^\ell \times 2^\ell$ -matrices and  $m_1, m'_1 \subseteq \bar{S}$ . Note also that we implicitly restricted the firing relation to the markings on  $\bar{S}$ .

► **Example 4.** In order to illustrate this, we go back to gossip diffusion (Fig. 1a, Ex. 1). Our input is the following: an initial probability distribution, describing the a priori knowledge, given by an MBN. Here we have no information about who knows or does not know the secret and hence we assume a uniform probability distribution over all markings. This is represented by the Bayesian network in Fig. 3a where each node is associated with a  $2 \times 1$ -matrix (vector)  $K_i$  where both entries are  $1/2$ .

Also part of the input is the family of transition distributions  $(r_n)_{n \in \mathbb{N}_0}$ . Here we assume that the firing probabilities of transitions are as in Example 1, but not all users are active at the same time. We have information that in the first step only users 1 and 2 are active, hence by normalization we obtain probabilities  $1/4, 1/2, 1/4$  for transitions  $d_1, d_2, d_3$  (the other transitions are deactivated).

<sup>4</sup> Without loss of generality we assume that the outputs have been permuted such that places in  $\bar{S}$  occur first in the sequence of places.



■ **Figure 3** Example: transformation of modular Bayesian networks.

Now we observe a success step. According to Sct. 3 we can make an update with  $P_*$  where  $P$  is the transition matrix of the Markov chain. Since none of the transitions is attached to place  $K_4$  the optimizations of this section allow us to represent  $P_*$  as  $P' \otimes \text{Id}_2$  where  $P'$  is an  $8 \times 8$ -matrix. E.g., as discussed in Ex. 1, we have  $P'(110 | 110) = 3/4$ ,  $P'(111 | 110) = 1/4$ . This matrix is simply attached to the modular Bayesian network (see Fig. 3b).

Now assume that it is our task to compute the probability that place  $K_3$  is marked. For this, we compute the corresponding marginal probabilities by terminating each output wire (apart from the third one) (see Fig. 3c). “Terminating a wire” means to remove it from the output wires. This results in summing up over all possible values assigned to each wire, where we can completely omit the last component, which is the unit of the Kronecker product. Note that the resulting vector is sub-stochastic and still has to be normalized. The normalization factor can be obtained by terminating also the remaining third wire, which gives us the probability mass of the sub-probability distribution. Our implementation will now tell us that place  $K_3$  is marked with probability  $5/8$ .

## 6 Variable Elimination and Tree Decompositions

### 6.1 Motivation

Given a modular Bayesian network, it is inefficient to obtain the full distribution, not just from the point of view of the computation, but also since its direct representation is of exponential size. However what we often need is to compute a marginal distribution (e.g., the probability that a certain place is marked) or a normalization factor for a sub-stochastic probability distribution (cf. Ex. 4). Another application would be to transform an MBN into an OBN, by isolating that part of the network that does not conform to the properties of an OBN, evaluating it and replacing it by an equivalent OBN.

Def. 7 gives a recipe for the evaluation, which is however quite inefficient. Hence we will now explain and adapt the well-known concept of variable elimination [14, 13]. Let us study the problem with a concrete example. Consider the Bayesian network  $B'$  in Fig. 2 and its evaluation described in Ex. 3. If we perform this computation one has to enumerate  $2^4 = 16$  bit vectors of length 4. Furthermore, after eliminating  $d$  we have to represent a matrix (also called *factor* in the literature on Bayesian networks) that is dependent on four random variables ( $a, b, c, e$ ), hence we say that it has width 4 ( $2^4 = 16$  entries).

However, it is not difficult to see that we can – via the distributive law – reorder the products and sums to obtain a more efficient way of computing the values:

$$M_{ev}(B')(e) = \sum_{d \in \{0,1\}} \left( \sum_{c \in \{0,1\}} \left( \sum_{b \in \{0,1\}} \left( \sum_{a \in \{0,1\}} (A(a) \cdot D(d | ab)) \cdot B(b) \right) \cdot C(c) \right) \cdot E(e | cd) \right).$$

In this way we obtain smaller matrices, the largest matrix (or factor) that occurs is  $D$  (width 3). Choosing a different elimination order might have been worse. For instance, if we had eliminated  $d$  first, we would have to deal with a matrix dependent on  $a, b, c, e$  (width 4).

## 6.2 Variable elimination

The literature of Bayesian networks [14, 13] extensively studies the best variable elimination order and discusses the relation to treewidth. For our setting we have to extend the results in the literature, since we also allow generators with more than one output.

► **Definition 9** (Elimination order). *Let  $B = (V, \ell, s, \text{out})$  be the causality graph of a modular Bayesian network of type  $n \rightarrow m$ . As in Def. 5 let  $W_B$  be the set of wires.*

*We define an undirected graph  $U_0$  that has as vertices<sup>5</sup> the wires  $W_B$  and two wires  $w_1, w_2$  are connected by an edge whenever they are connected to the same node. More precisely, they are connected whenever they are input or output wires for the same node (i.e.  $w_1, w_2$  are both in  $s(v)t(v)$  for a node  $v \in V$ ).*

*Now let  $w_1, \dots, w_k$  (where  $k = |W_B|$ ) be an ordering of the internal wires, a so-called elimination ordering. We update the graph  $U_{i-1}$  to  $U_i$  by removing the next wire  $w_i$  and connecting all of its neighbours by edges (so-called fill in). External wires are never eliminated. The width of the elimination ordering is the size of the largest clique that occurs in some graph  $U_i$ . The elimination width of  $B$  is the least width taken over all orderings.*

In the case of Bayesian networks, the set of wires of an OBN corresponds to the set of random variables. In the literature, the graph  $U_0$  is called the moralisation of the Bayesian network, it is obtained by taking the Bayesian network (an acyclic graph), forgetting about the direction of the edges, and connecting all the parents (i.e., the predecessors) of a random variable, i.e. making them form a clique. This results in the same graph as the construction described above.

To introduce the algorithm, we need the notion of a *factor*, already hinted at earlier.

► **Definition 10** (Factor). *Let  $(B, ev)$  be a modular Bayesian network with a set of wires  $W_B$ . A factor  $(f, \tilde{w})$  of size  $s$  consists of a map  $f: \{0, 1\}^s \rightarrow [0, 1]$  together with a sequence of wires  $\tilde{w} \in W_B^*$ . We require that  $\tilde{w}$  is of length  $s$  ( $|\tilde{w}| = s$ ) and does not contain duplicates.*

*Given a wire  $w \in W_B$  and a multiset  $\mathcal{F}$  of factors, we denote by  $C_w(\mathcal{F})$  all those factors  $(f, \tilde{w}) \in \mathcal{F}$  where  $\tilde{w}$  contains  $w$ . By  $X_w(\mathcal{F})$  we denote the set of all wires that occur in the factors in  $C_w(\mathcal{F})$ , apart from  $w$ .*

We now consider an algorithm that computes the probability distribution represented by a modular Bayesian network of type  $n \rightarrow m$ . We assume that an evaluation map  $ev$ , mapping generators to their corresponding matrices, and an elimination order  $w_1, \dots, w_k$  of internal wires is given. Furthermore, given a sequence of wires  $\tilde{w} = w'_1 \dots w'_s$  and a bitstring  $\mathbf{x} = x_1 \dots x_s$ , we define the substitution function  $b_{\tilde{w}, \mathbf{x}}$  from wires to bits as  $b_{\tilde{w}, \mathbf{x}}(w'_j) = x_j$ .

► **Algorithm 11** (Variable elimination).

Input: An MBN  $(B, ev)$  of type  $n \rightarrow m$

- Let  $\mathcal{F}_0$  be the initial multiset of factors. For each node  $v$  of type  $n_{\ell(v)} \rightarrow m_{\ell(v)}$ , it contains the matrix  $ev(v)$ , represented as a factor  $f$ , together with the sequence  $s(v)t(v)$ . That is  $f(\mathbf{xy}) = ev(v)(\mathbf{y} \mid \mathbf{x})$  where  $\mathbf{x} \in \{0, 1\}^{n_{\ell(v)}}$ ,  $\mathbf{y} \in \{0, 1\}^{m_{\ell(v)}}$ .

<sup>5</sup> We talk about the *nodes* of an MBN  $B$  and the *vertices* of an undirected graph  $U_i$ .

## 38:12 Uncertainty Reasoning for Probabilistic Petri Nets via Bayesian Networks

- Now assume that we have a set  $\mathcal{F}_{i-1}$  of factors and take the next wire  $w_i$  in the elimination order. We choose all those factors that contain  $w_i$  and compute a new factor  $(f, \tilde{w})$ . Let  $\tilde{w}$  be a sequence that contains all wires of  $X_w(\mathcal{F}_{i-1})$  (in arbitrary order, but without duplicates). Let  $s = |\tilde{w}|$ . Then  $f$  is a function of type  $f: \{0, 1\}^s \rightarrow [0, 1]$ , defined as:

$$f(\mathbf{y}) = \sum_{z \in \{0, 1\}} \prod_{(g, \tilde{w}^g) \in C_{w_i}(\mathcal{F}_{i-1})} g(b_{\tilde{w}w_i, \mathbf{yz}}(\tilde{w}^g)).$$

We set  $\mathcal{F}_i = \mathcal{F}_{i-1} \setminus C_{w_i}(\mathcal{F}_{i-1}) \cup \{(f, \tilde{w})\}$ .

- After the elimination of all wires we obtain a multiset of factors  $\mathcal{F}_k$ , whose sequences contain only input and output wires. The resulting probability distribution is  $p: \{0, 1\}^{n+m} \rightarrow [0, 1]$ , where  $\mathbf{x} \in \{0, 1\}^n$ ,  $\mathbf{y} \in \{0, 1\}^m$ ,  $\tilde{i} = i_1 \dots i_n$ ,  $\tilde{o} = \text{out}(o_1) \dots \text{out}(o_m)$ :

$$p(\mathbf{xy}) = \prod_{(f, \tilde{w}^f) \in \mathcal{F}_k} f(b_{\tilde{i}\tilde{o}, \mathbf{xy}}(\tilde{w}^f))$$

That is, given the next wire  $w_i$  we choose all factors that contain this wire, remove them from  $\mathcal{F}_{i-1}$  and multiply them, while eliminating the wire. The next set is obtained by adding the new factor. Finally, we have factors that contain only input and output wires and we obtain the final probability distribution by multiplying them.

► **Proposition 12.** *Given a modular Bayesian network  $(B, ev)$  where  $B$  is of type  $n \rightarrow m$ , Algorithm 11 computes its corresponding (sub-)stochastic matrix  $M_{ev}(B)$ , that is*

$$M_{ev}(B)(\mathbf{y} \mid \mathbf{x}) = p(\mathbf{xy}) \quad \text{for } \mathbf{x} \in \{0, 1\}^n, \mathbf{y} \in \{0, 1\}^m.$$

Furthermore, the size of the largest factor in any multiset  $\mathcal{F}_i$  is bounded by the width of the elimination ordering.

### 6.3 Comparison to Treewidth

We conclude this section by investigating the relation between elimination width and the well-known notion of treewidth [2].

► **Definition 13** (Treewidth of a causality graph). *Let  $B = (V, \ell, s, \text{out})$  be a causality graph of type  $n \rightarrow m$ . A tree decomposition for  $B$  is an undirected tree  $T = (V_T, E_T)$  such that*

- every node  $t \in V_T$  is associated with a bag  $X_t \subseteq W_B$ ,
- every wire  $w \in W_B$  is contained in at least one bag  $X_t$ ,
- for every node  $v \in V$  there exists a bag  $X_t$  such that all input and output wires of  $v$  are contained in  $X_t$  (i.e., all wires in  $s(v)$  and  $t(v)$  are in  $X_t$ ) and
- for every wire  $w \in W_B$ , the tree nodes  $\{t \in V_T \mid w \in X_t\}$  form a subtree of  $T$ .

The width of a tree decomposition is given by  $\max_{t \in V_T} |X_t| - 1$ .

The treewidth of  $B$  is the minimal width, taken over all tree decompositions.

Note that the treewidth of a causality graph corresponds to the treewidth of the graph  $U_0$  from Def. 9. Now we are ready to compare elimination width and treewidth.

► **Proposition 14.** *Elimination width is always an upper bound for treewidth and they coincide when  $B$  is a causality graph of type  $0 \rightarrow 0$ . For a network of type  $0 \rightarrow m$  the treewidth may be strictly smaller than the elimination width.*

The treewidth might be strictly smaller since we are now allowed to eliminate output wires. However, it is easy to see that the treewidth plus the number of output wires always provides an upper bound for the elimination width.

The paper [2] also discusses heuristics for computing good elimination orderings, an optimization problem that is NP-hard. Hence the treewidth of a causality graph gives us an upper bound for the most costly step in computing its corresponding probability distribution. [26] shows that a small treewidth is actually a necessary condition for obtaining efficient inference algorithms.

We can also compare elimination width to the related notion of term width, more details can be found in [1].

## 7 Implementation and Runtime Results

We extended the implementation presented in the predecessor paper [5] by incorporating probabilistic Petri nets and elimination orderings, in order to evaluate the performance of the proposed concepts. The implementation is open source and freely available from GitHub.<sup>6</sup>

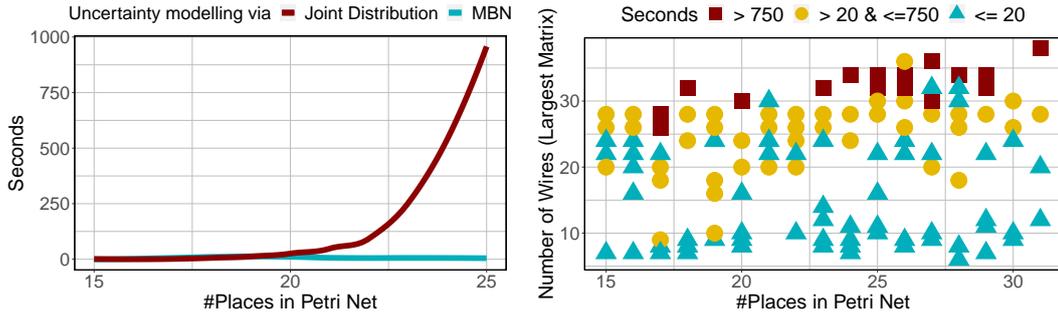
Runtime results were obtained by randomly generating Petri nets with different parameters, e.g. number of places, transitions and tokens, initial marking. The maximal number of places in pre- and post-conditions is restricted to three and at most five transitions are enabled in each step. With these parameters, the worst case scenario is the creation of a matrix of type  $30 \rightarrow 30$ . After the initialization of a Petri net, which can be interpreted with either semantics (independent/stochastic), transitions and their probabilities are picked at random. Then we observe either success or failure and update the probability distribution accordingly.

We select the elimination order via a heuristics by preferring wires with minimal degree in the graph  $U_i$  (cf. Def 9). Furthermore we apply a few optimizations: Nodes with no output wires will be evaluated first, nodes without inputs second. The observation of a failure will generate a diagonal matrix, which enables an optimized evaluation, as its input and output wires have to carry the same value (otherwise we obtain a factor 0). In addition, we use optimizations whenever we have definitive knowledge about the marking of a particular place (of a pre-condition), by drawing conclusions about the ability to fire certain transitions.

The plot on the left of Fig. 4 compares runtimes when incorporating ten success/failure observations directly on the joint distribution (i.e. the naive representation of a probability distribution) versus our MBN implementation. We initially assume a uniform distribution of tokens and calculate the probability that the first place is marked after the observations. Both approaches evaluate the same Petri net and therefore calculate the same results. The data is for the independent semantics, but it is very similar for the stochastic semantics.

While the runtime increases exponentially when using joint distributions, our MBN implementation stays relatively constant (see Fig. 4, left). Due to memory issues, handling Petri nets with more than 30 places is not anymore feasible for the direct computation of joint distributions. We use the median for comparison (see Fig. 4, left), but if an MBN consists of very large matrices, the evaluation time will be rather high. The right plot of Fig. 4 shows this correlation, where colours denote the runtime and the  $y$ -axis represents the number of wires attached to the largest matrix. (Here we actually count equivalence classes by grouping those wires that have to carry the same value, due to their attachment to a diagonal matrix, see also the optimization explained above.)

<sup>6</sup> <https://github.com/RebeccaBe/Bayesian-II>



■ **Figure 4** Left: Median of runtimes performing after 10 transitions on a Petri net. Right: Effect of large matrices on the runtimes of the MBN implementation.

The advantage of our approach decreases when we have substantially more places in the pre- and post-set, more transitions that may fire and a larger number of steps, since then the Bayesian network is more densely connected and contains larger matrices. Furthermore, one might generally expect the state (containing tokens or not) of places of the Petri net to become more and more coupled over time, as more transitions have fired, decreasing the performance improvement we gain from using MBNs. However, recall that the transitions that can fire at any time are explicitly controlled by the input  $p_T^a$ . This allows our model to capture situations where different parts of the network stay uncoupled over time and where using MBNs is an advantage. Furthermore the observation of a failure allows an optimized variable elimination, as explained above.

## 8 Conclusion

We propose a framework for uncertainty reasoning for probabilistic Petri nets that represents probability distributions compactly via Bayesian networks. In particular we describe how to efficiently update and evaluate Bayesian networks.

*Related work:* Naturally, uncertainty reasoning has been considered in many different scenarios (for an overview see [19]). Here we review only those approaches that are closest to our work.

In [5] we studied a simpler scenario for nets whose transitions do not fire probabilistically, but are picked by the observer, resulting in a restricted set of update operations. Rather than computing marginal distributions directly via variable elimination as in this paper, our aim there was to transform the resulting modular Bayesian network into an ordinary one. Since the updates to the net were of a simpler nature, we were able to perform this conversion. Here we are dealing with more complex updates where this can not be done efficiently. Instead we are concentrating on extracting information, such as marginal distributions, from a Bayesian network.

Furthermore, uncertainty reasoning as described in Sect. 3 is related to the methods used for hidden Markov models [32], where the observations refer to the states, whereas we (partially) observe the transitions.

There are several proposals which enrich Petri nets with a notion of uncertainty: possibilistic Petri nets [27], plausible Petri nets [9] that combine discrete and continuous processes or fuzzy Petri nets [6, 34] where firing of transitions is governed by the truth values of statements. Uncertainty in connection with Petri nets is also treated in [25, 23], but without introducing a formal model. As far as we know neither approach considers symbolic representation of probability distributions via Bayesian networks.

In [3] the authors exploit the fact that Petri nets also have a monoidal structure and describe how to convert an occurrence (Petri) net with a truly concurrent semantics into a Bayesian network, allowing to derive probabilistic information, for instance on whether a place will eventually be marked. This is different from our task, but it will be interesting to compare further by unfolding our nets and equipping them with a truly concurrent semantics, based on the probabilistic information from the time-inhomogeneous Markov chain.

We instead propose to use Bayesian networks as symbolic representations of probability distributions. An alternative would be to employ multi-valued (or multi-terminal) binary decision diagrams (BDDs) as in [20]. An exact comparison of both methods is left for future work. We believe that multi-valued BDDs will fare better if there are only few different numerical values in the distribution, otherwise Bayesian networks should have an advantage.

As mentioned earlier, representing Bayesian networks by PROPs or string diagrams is a well-known concept, see for instance [16, 22]. The paper [21] describes another transformation of Bayesian networks by string diagram surgery that models the effect of an intervention.

In addition there is a notion of dynamic Bayesian networks [30], where a random variable has a separate instance for each time slice. We instead keep only one instance of every random variable, but update the Bayesian network itself.

In addition to variable elimination, a popular method to compute marginals of a probability distribution is based on belief propagation and junction trees [28]. In order to assess the potential efficiency gain, this approach has to be adapted for modular Bayesian networks. However due to the dense interconnection and large matrices of MBNs, an improvement in runtime is unclear and deserves future investigation.

*Future work:* One interesting avenue of future work is to enrich our model with timing information by considering continuous-time Markov chains [35], where firing delays are sampled from an exponential distribution. Instead of asking about the probability distribution after  $n$  steps we could instead ask about the probability distribution at time  $t$ .

We would also like to add mechanisms for controlling the system, such as transitions that are under the control of the observer and can be fired whenever enabled. Then the task of the observer would be to control the system and guide it into a desirable state. In this vein we are also interested in studying stochastic games [11] with uncertainty.

The interaction between the structure of the Petri net and the efficiency of the analysis method also deserves further study. For instance, are free-choice nets [15] – with restricted conflicts of transitions – more amenable to this type of analysis than arbitrary nets?

Recently there has been a lot of interest in modelling compositional systems via string diagrams, in the categorical setting of symmetric monoidal categories or PROPs [10]. In this context it would be interesting to see how the established notion of treewidth [2] and its algebraic characterizations [12] translates into a notion of width for string diagrams. We started to study this for the notion of term width, but we are not aware of other approaches, apart from [7] which considers monoidal width.

---

## References

- 1 Rebecca Bernemann, Benjamin Cabrera, Reiko Heckel, and Barbara König. Uncertainty reasoning for probabilistic petri nets via Bayesian networks, 2020. arXiv:2009.14817. URL: <https://arxiv.org/abs/2009.14817>.
- 2 H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations I. Upper bounds. Technical Report UU-CS-2008-032, Department of Information and Computing Sciences, Utrecht University, September 2008.

- 3 R. Bruni, H. C. Melgratti, and U. Montanari. Bayesian network semantics for Petri nets. *Theoretical Computer Science*, 807:95–113, 2020.
- 4 B. Cabrera. *Analyzing and Modeling Complex Networks – Patterns, Paths and Probabilities*. PhD thesis, Universität Duisburg-Essen, 2019.
- 5 B. Cabrera, T. Heindel, R. Heckel, and B. König. Updating probabilistic knowledge on Condition/Event nets using Bayesian networks. In *Proc. of CONCUR '18*, volume 118 of *LIPICs*, pages 27:1–27:17. Schloss Dagstuhl – Leibniz Center for Informatics, 2018. URL: [http://drops.dagstuhl.de/opus/frontdoor.php?source\\_opus=9565](http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=9565).
- 6 J. Cardoso, R. Valette, and D. Dubois. Fuzzy Petri nets: An overview. In *Proc. of 13th Triennial World Congress*, 1996.
- 7 A. Chantawibul and P. Sobociński. Towards compositional graph theory. In *Proc. of MFPS XXXI*. Elsevier, 2015. ENTCS 319.
- 8 E. Charniak. Bayesian networks without tears. *AI magazine*, 12(4):50–50, 1991.
- 9 M. Chiachio, J. Chiachio, D. Prescott, and J.D. Andrews. A new paradigm for uncertain knowledge representation by plausible Petri nets. *Information Sciences*, 453:323–345, 2018.
- 10 B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.
- 11 A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.
- 12 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic, A Language-Theoretic Approach*. Cambridge University Press, June 2012.
- 13 A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2011.
- 14 R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
- 15 J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
- 16 B. Fong. Causal theories: A categorical perspective on Bayesian networks. Master’s thesis, University of Oxford, 2012. arXiv:1301.6201.
- 17 N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.
- 18 C.M. Grinstead and J.L. Snell. *Introduction to probability*. American Mathematical Soc., 2012.
- 19 J.Y. Halpern. *Reasoning about Uncertainty*. MIT Press, second edition edition, 2017.
- 20 H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous-time markov chains. In *Proc. of NSMC '99 (International Workshop on the Numerical Solution of Markov Chains)*, pages 188–207, 1999.
- 21 B. Jacobs, A. Kissinger, and F. Zanasi. Causal inference by string diagram surgery. In *Proc. of FOSSACS '19*, pages 313–329. Springer, 2019. LNCS 11425.
- 22 B. Jacobs and F. Zanasi. A formal semantics of influence in Bayesian reasoning. In *Proc. of MFCS*, volume 83 of *LIPICs*, pages 21:1–21:14, 2017.
- 23 I. Jarkass and M. Rombaut. Dealing with uncertainty on the initial state of a Petri net. In *Proc. of UAI '98 (Uncertainty in Artificial Intelligence)*, pages 289–295, 1998.
- 24 M.J. Keeling and K.T.D. Eames. Networks and epidemic models. *Journal of the Royal Society Interface*, 2(4):295–307, 2005.
- 25 M. Kuchárik and Z. Balogh. Modeling of uncertainty with Petri nets. In *Proc. of ACIIDS '19 (Asian Conference on Intelligent Information and Database Systems)*, pages 499–509. Springer, 2019. LNAI 11431.
- 26 J.H.P. Kwisthout, H.L. Bodlaender, and L.C. Van Der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *Proc. of ECAI '10 (European Conference on Artificial Intelligence)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 237–242. IOS Press, 2010.

- 27 J. Lee, K.F.R. Liu, and W. Chiang. Modeling uncertainty reasoning with possibilistic Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 33(2):214–224, 2003.
- 28 V. Lepar and P.P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In G.F. Cooper and S. Moral, editors, *Proc. of UAI '98 (Uncertainty in Artificial Intelligence)*, pages 328–337, 1998. URL: <http://arxiv.org/abs/1301.7394>.
- 29 M. Ajmone Marsan. Stochastic Petri nets: an elementary introduction. In *Proc. of the European Workshop on Applications and Theory in Petri Nets*, volume 424 of *Lecture Notes in Computer Science*, pages 1–29. Springer, 1990.
- 30 K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.
- 31 J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proc. of the 7th Conference of the Cognitive Science Society*, pages 329–334, 1985. UCLA Technical Report CSD-850017.
- 32 L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- 33 W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1985.
- 34 Z. Suraj. Generalised fuzzy Petri nets for approximate reasoning in decision support systems. In *Proc. of CS&P '12 (International Workshop on Concurrency, Specification and Programming)*, volume 928 of *CEUR Workshop Proceedings*, pages 370–381. CEUR-WS.org, 2012.
- 35 A. Tolver. An introduction to Markov chains. Department of Mathematical Sciences, University of Copenhagen, November 2016.
- 36 W. Wiegner, W. Burgers, and B. Kappen. Bayesian networks, introduction and practical applications. In *Handbook on Neural Information Processing*, pages 401–431. Springer, 2013.



# Synthesizing Safe Coalition Strategies

Nathalie Bertrand 

Université Rennes, Inria, CNRS, IRISA, Rennes, France

Patricia Bouyer 

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Gif-sur-Yvette, France

Anirban Majumdar

Université Rennes, Inria, CNRS, IRISA, Rennes, France

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Gif-sur-Yvette, France

---

## Abstract

Concurrent games with a fixed number of agents have been thoroughly studied, with various solution concepts and objectives for the agents. In this paper, we consider concurrent games with an arbitrary number of agents, and study the problem of synthesizing a coalition strategy to achieve a global safety objective. The problem is non-trivial since the agents do not know *a priori* how many they are when they start the game. We prove that the existence of a safe arbitrary-large coalition strategy for safety objectives is a PSPACE-hard problem that can be decided in exponential space.

**2012 ACM Subject Classification** Theory of computation → Verification by model checking

**Keywords and phrases** concurrent games, parameterized verification, strategy synthesis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.39

**Related Version** <https://arxiv.org/abs/2008.03770>.

## 1 Introduction

**Context.** The generalisation and everyday usage of modern distributed systems call both for the verification and synthesis of algorithms or strategies running on distributed systems. Concrete examples are cloud computing, blockchain technologies, servers with multiple clients, wireless sensor networks, bio-chemical systems, or fleets of drones cooperating to achieve a common goal [11]. In their general form, these systems are not only distributed, but they may also involve an arbitrary number of agents. This explains the interest of the model-checking community both for the verification of parameterized systems [15, 9], and for the synthesis of distributed strategies [21]. Our contribution is at the crossroad of those topics.

**Parameterized verification.** Parameterized verification refers here to the verification of systems formed of an arbitrary number of agents. Often, the precise number of agents is unknown, yet, algorithms and protocols running on such distributed systems are designed to operate correctly independently of the number of agents. The automated verification and control of crowds, *i.e.*, in case the agents are anonymous, is challenging. Remarkably, subtle changes, such as the presence or absence of a controller in the system, can drastically alter the complexity of the verification problems [15]. In the decidable cases, the intuition that bugs appear for a small number of agents is sometimes confirmed theoretically by the existence of a cutoff property, which reduces the parameterized model checking to the verification of finitely many instances [14]. In the last 15 years, parameterised verification algorithms were successfully applied to *e.g.*, cache coherence protocols in uniform memory access multiprocessors [13], or the core of simple reliable broadcast protocols in asynchronous systems [17]. When agents have unique identifiers, most verification problems become undecidable, especially if one can use identifiers in the code agents execute [3].



© Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 39; pp. 39:1–39:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To our knowledge, there are few works on controlling parameterized systems. Exceptions are, control strategies for (probabilistic) broadcast networks [7] and for crowds of (probabilistic) automata [6, 18, 12].

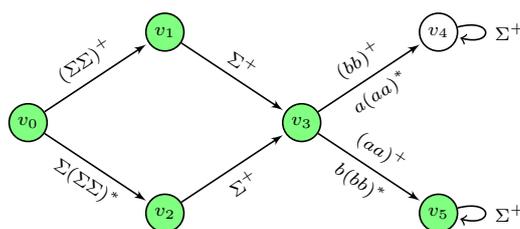
**Distributed synthesis.** The problem of distributed synthesis asks whether strategies for individual agents can be designed to achieve a global objective, in a context where individuals have only a partial knowledge of the environment. There are several possible formalizations for distributed synthesis, for instance via an architecture of processes with communication links between agents [21], or using coordination games [20, 19, 8]. The two settings are linked, and many (un)decidability results have been proven, depending on various parameters.

**Concurrent games on graphs.** By allowing complex interactions between agents, concurrent games on graphs [1, 2] are a model of choice in several contexts, for instance for multi-agents systems, or for coordination or planning problems. An arena for  $n$  agents is a directed graph where the transitions are labeled by  $n$ -tuples of actions (or simply words of length  $n$ ). At each vertex of the graph, all  $n$  agents select simultaneously and independently an action, and the next vertex is determined by the combined move consisting of all the actions (or word formed of all the actions). Most often, one considers infinite duration plays, *i.e.*, plays generated by iterating this process forever. Concepts studied on multiagent concurrent games include many borrowed from game theory, such as winning strategies (see *e.g.*, [1]), rationality of the agents (see *e.g.*, [16]), Nash equilibria (see *e.g.*, [23, 10]).

**Parameterized concurrent games on graphs.** In a previous work, we introduced concurrent games in which the number of agents is arbitrary [4]. These games generalize concurrent games with a fixed number of agents, and can be seen as a succinct representation of infinitely many games, one for each fixed number of agents. This is done by replacing, on edges of the arena, words representing the choice of each of the agents by languages of finite yet *a priori* unbounded words. Such a parameterized arena can represent infinitely many interaction situations, one for each possible number of agents. In parameterized concurrent games, the agents do not know *a priori* the number of agents participating to the interaction. Each agent observes the action it plays and the vertices the play goes through. These pieces of information may refine the knowledge each agent has on the number of involved agents.

Such a game model raises new interesting questions, since the agents do not know beforehand how many they are. In [4], we first considered the question of whether Agent 1 can ensure a reachability objective independently of the number of her opponents, and no matter how they play. The problem is non trivial since Agent 1 must win with a *uniform* strategy. We proved that when edges are labeled with regular languages, the problem is PSPACE-complete; and for positive instances one can effectively compute a winning strategy in polynomial space.

**Contribution.** In this paper, we are interested in the coordination problem in concurrent parameterized games, with application to distributed synthesis. Given a game arena and an objective, the problem consists in synthesizing for every potential agent involved in the game a strategy that she should apply, so that, collectively, a global objective is satisfied. In our setting, it is implicit that agents have identifiers. However agents do not communicate; their identifier will only be used to select the vertices the game proceeds to. Furthermore, agents do not know how many they are, they only see vertices which are visited, and can infer information about the number of agents involved in the game.



■ **Figure 1** Example of a parameterized arena. All unspecified transitions lead to vertex  $v_4$ .

To better understand the model and the problem, consider the game arena depicted on Fig. 1. Edges are labeled by (regular) languages. Assuming the game starts at  $v_0$ , the game proceeds as follows: a positive integer  $k$  is selected by the environment, but is not revealed to the agents; then an infinite word  $w \in \Sigma^\omega$  is selected collectively by the agents (this is the *coalition strategy*); the  $n$ -th letter of  $w$  represents the action played by Agent  $n$ ;<sup>1</sup> depending on whether the prefix of length  $k$  of  $w$  belongs to  $(\Sigma\Sigma)^+$  (in case  $k$  is even) or  $\Sigma(\Sigma\Sigma)^*$  (in case  $k$  is odd), the game proceeds to vertex  $v_1$  or  $v_2$ ; the process is repeated ad infinitum, generating an infinite play in the graph. Depending on the winning condition, the play will be winning or losing for  $k$ . The coalition strategy will be said winning whenever the generated play is winning whatever the selected number  $k$  of agents is.

In this example, assuming the winning condition is to stay in the green vertices, there is a simple winning strategy: play  $a^\omega$  in  $v_0, v_1$  and  $v_2$  (that is, all agents should play an  $a$ ), and if the game has gone through  $v_1$  (case of an even number of agents), then play  $a^\omega$  in  $v_3$  (all agents should play an  $a$ ), otherwise play  $b^\omega$  in  $v_3$  (all agents should play a  $b$ ). This ensures that the play never ends up in vertex  $v_4$ .

In this paper, we focus on safety winning conditions: the agents must collectively ensure that only safe vertices are visited along any play compatible with the coalition strategy in the game. We prove that the existence of a winning coalition strategy is decidable in exponential space, and that it is a PSPACE-hard problem. For positive instances, winning coalition strategies with an exponential-size memory structure can be synthesized in exponential space.

## 2 Game setting

We use  $\mathbb{N}_{>0}$  for the set of positive natural numbers. For an alphabet  $\Sigma$  and  $k \in \mathbb{N}_{>0}$ ,  $\Sigma^k$  denotes the set of all finite words of length  $k$ ,  $\Sigma^+$  denotes the set of all finite but non-empty words, and  $\Sigma^\omega$  denotes the set of all infinite words. For two words  $u \in \Sigma^+$  and  $w \in \Sigma^+ \cup \Sigma^\omega$ , we write  $u \sqsubseteq w$  to denote  $u$  is a prefix of  $w$ , and for any  $k \in \mathbb{N}_{>0}$ ,  $[w]_{\leq k}$  denotes the prefix of length  $k$  of  $w$  (belongs to  $\Sigma^k$ ).

We introduced parameterized arenas in [4], a model of arenas with a parameterized number of agents. Parameterized arenas extend arenas for concurrent games with a fixed number of agents [1], by labeling the edges with languages over finite words, which may be of different lengths. Each word represents a joint move of the agents, for instance  $u = a_1 \cdots a_k \in \Sigma^k$  assumes there are  $k$  agents, and for every  $1 \leq n \leq k$ , Agent  $n$  chooses action  $a_n$ .

► **Definition 1.** A parameterized arena is a tuple  $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$  where

- $V$  is a finite set of vertices;

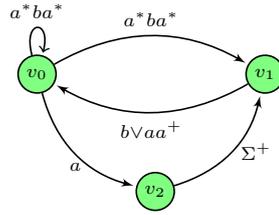
<sup>1</sup> This is where identifiers are implicitly used.

## 39:4 Synthesizing Safe Coalition Strategies

- $\Sigma$  is a finite set of actions;
- $\Delta : V \times V \rightarrow 2^{\Sigma^+}$  is a partial transition function.

It is required that for every  $(v, v') \in V \times V$ ,  $\Delta(v, v')$  describes a regular language.

Fix a parameterized arena  $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$ . The arena  $\mathcal{A}$  is *deterministic* if for every  $v \in V$ , and every word  $u \in \Sigma^+$ , there is at most one vertex  $v' \in V$  such that  $u \in \Delta(v, v')$ . The arena is assumed to be *complete*: for every  $v \in V$  and  $u \in \Sigma^+$ , there exists  $v' \in V$  such that  $u \in \Delta(v, v')$ . This assumption is natural: such an arena will be used to play games with an arbitrary number of agents, hence for the game to be non-blocking, successor vertices should exist whatever that number is and irrespective of the choices of actions.



■ **Figure 2** Example of a non-deterministic parameterized arena. Only safe vertices (colored in green) have been depicted here. All unspecified transitions lead to a non-safe vertex  $\perp$ .

► **Example 2.** We already gave an example in the introduction. Let us give another example, which will be useful for illustrating the constructions made in the paper. Fig. 2 presents a non-deterministic parameterized arena. As such the arena is not complete, we assume that all unspecified moves lead to an extra losing vertex  $\perp$ , not depicted here. If for some number of agents  $k$  (selected by environment and not known to the agents), the  $k$ -length prefix of the word collectively chosen by the agents at  $v_0$  belongs to  $a^*ba^*$ , then the play either stays at  $v_0$  or moves to  $v_1$  (again selected by environment).

**History, play and strategy.** We fix a parameterized arena  $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$ . A *history* in  $\mathcal{A}$  is a finite sequence of vertices, that is compatible with the edges: formally,  $h = v_0v_1 \dots v_p \in V^+$  such that for every  $1 \leq j < p$ ,  $\Delta(v_j, v_{j+1})$  is defined. We write  $\text{Hist}_{\mathcal{A}}$  for the set of all histories. An infinite sequence of vertices compatible with the edges is called a *play*.

A *strategy* for Agent  $n$  is a mapping  $\sigma_n : \text{Hist}_{\mathcal{A}} \rightarrow \Sigma$  that associates an action to every history. A *strategy profile* is a tuple of strategies, one for each agent. Since the number of agents is not fixed *a priori*, a strategy profile is an infinite tuple of strategies:  $\tilde{\sigma} = \langle \sigma_1, \sigma_2, \dots \rangle = (\text{Hist}_{\mathcal{A}} \rightarrow \Sigma)^\omega$ .

■ **Table 1** From strategy profile to coalition strategy.

	$h_0$	$h_1$	$h_2$	$h_3$	$\dots$
$\sigma_1$	$a$	$b$	$b$	$b$	$\dots$
$\sigma_2$	$b$	$b$	$b$	$b$	$\dots$
$\sigma_3$	$b$	$a$	$a$	$a$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Observe that a strategy profile can equivalently be described as a *coalition strategy*  $\sigma : \text{Hist}_{\mathcal{A}} \rightarrow \Sigma^\omega$ , as illustrated in Table 1. Indeed, if an enumeration of histories  $(h_j)_{j \in \mathbb{N}}$  is fixed, a strategy profile can be seen as a table with infinitely many rows –one for each agent–

and infinitely many columns indexed by histories. Reading the table vertically provides the coalition strategy view: each history is mapped to an  $\omega$ -word, obtained by concatenating the actions chosen by each of the agents. Since, in this paper, we are interested in the existence of a winning strategy profile, it is equivalent to asking the existence of a winning coalition strategy (they may not be equivalent for some other decision problems). In the sequel, we mostly take the coalition strategy view, but may interchangeably also consider strategy profiles.

**Finite memory coalition strategies.** Let  $\sigma : \text{Hist}_{\mathcal{A}} \rightarrow \Sigma^\omega$  be a coalition strategy and  $\mathbb{M}$  be a set. We say that the strategy  $\sigma$  uses memory  $\mathbb{M}$  whenever there exist  $\mathfrak{m}_{\text{init}} \in \mathbb{M}$  and applications  $\text{upd} : \mathbb{M} \times V \rightarrow \mathbb{M}$  and  $\text{act} : \mathbb{M} \times V \rightarrow \Sigma^\omega$  such that by defining inductively  $\mathfrak{m}[h] \in \mathbb{M}$  by  $\mathfrak{m}[v_0] = \mathfrak{m}_{\text{init}}$  and  $\mathfrak{m}[h \cdot v] = \text{upd}(\mathfrak{m}[h], v)$ , we have that for every  $h \in \text{Hist}_{\mathcal{A}}$ ,  $\sigma(h) = \text{act}(\mathfrak{m}[h], \text{last}(h))$ , where  $\text{last}(h)$  is the last vertex of history  $h$ . The structure  $(\mathbb{M}, \text{upd})$  records information on the history seen so far ( $\mathfrak{m}[h]$  is the memory state “reached” after history  $h$ ), and  $\text{act}$  dictates how all the agents should play.

If  $\mathbb{M}$  is finite, then  $\sigma$  is said *finite-memory*, and if  $\mathbb{M}$  is a singleton, then  $\sigma$  is said *memoryless* (each choice only depends on the last vertex of the history).

**Realizability and outcomes.** For  $k \in \mathbb{N}_{>0}$ , we say a history  $h = v_0 \cdots v_p$  is *k-realizable* if it corresponds to a history for  $k$  agents, *i.e.*, if for all  $j < p$ , there exists  $u \in \Sigma^k$  with  $u \in \Delta(v_j, v_{j+1})$ . A history is *realizable* if it is *k-realizable* for some  $k \in \mathbb{N}_{>0}$ . Similarly to histories for finite sequences of consecutive vertices, one can define the notions of *(k-)realizable plays* for infinite sequences.

Given a coalition strategy  $\sigma$ , an initial vertex  $v_0$  and a number of agents  $k \in \mathbb{N}_{>0}$ , we define the *k-outcome*  $\text{Out}_{\mathcal{A}}^k(v_0, \sigma)$  as the set of all *k-realizable* plays induced by  $\sigma$  from  $v_0$ . Formally,  $\text{Out}_{\mathcal{A}}^k(v_0, \sigma) = \{v_0 v_1 \cdots \mid \forall j \in \mathbb{N}_{>0}, [\sigma(v_0 \cdots v_j)]_{\leq k} \in \Delta(v_j, v_{j+1})\}$ . Note that the completeness assumption ensures that the set  $\text{Out}_{\mathcal{A}}^k(v_0, \sigma)$  is not empty. Then the *outcome* of coalition strategy  $\sigma$  is simply  $\text{Out}_{\mathcal{A}}(v_0, \sigma) = \bigcup_{k \in \mathbb{N}_{>0}} \text{Out}_{\mathcal{A}}^k(v_0, \sigma)$ .

**The safety coalition problem.** We are now in a position to define our problem of interest. Given an arena  $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$ , a set of *safe* vertices  $S \subseteq V$  defines a *parameterized safety game*  $\mathcal{G} = (\mathcal{A}, S)$ . Without loss of generality we assume from now that  $V \setminus S$  are sinks. A coalition strategy  $\sigma$  from  $v_0$  in the safety game  $\mathcal{G} = (\mathcal{A}, S)$  is said *winning* if all induced plays only visit vertices from  $S$ :  $\text{Out}_{\mathcal{A}}(v_0, \sigma) \subseteq S^\omega$ . Our goal is to study the decidability and complexity of the existence of winning coalition strategies, and to synthesize such winning coalition strategies when they exist. We therefore introduce the following decision problem:

**SAFETY COALITION PROBLEM**

**Input:** A parameterized safety game  $\mathcal{G} = (\mathcal{A}, S)$  and an initial vertex  $v_0$ .

**Question:** Does there exist a coalition strategy  $\sigma$  such that  $\text{Out}_{\mathcal{A}}(v_0, \sigma) \subseteq S^\omega$ ?

The safety coalition problem is a coordination problem: agents should agree on a joint strategy which, when played in the graph and no matter how many agents are involved, the resulting play is safe. Note that, due to the link between coalition strategies and tuples of individual strategies mentioned on Page 4, the coalition strategies are distributed: the only information required for an agent to play her strategy is the history so far, not the number of agents selected by the environment; however she can infer some information about the number of agents from the history; this is for instance the case at vertex  $v_3$  in the example of Fig. 2. Note that there is no direct communication between agents.

► **Example 3.** We have already given in the introduction a winning coalition strategy for the game in Fig. 1. On the arena in Fig. 2, assuming  $\perp$  is the only unsafe vertex, one can also show that the agents have a winning coalition strategy  $\sigma$  from  $v_0$  to stay within green (*i.e.*, safe) vertices. Consider the coalition strategy  $\sigma$  such that  $\sigma(v_0) = aba^\omega$ ,  $\sigma(v_0v_2) = a^\omega$ ,  $\sigma(v_0v_1) = a^\omega$ , and  $\sigma(v_0v_2v_1) = b^\omega$ . Intuitively, on playing  $aba^\omega$  from  $v_0$ , in one step, the game either stays in  $v_0$  (which is “safe”) or moves to  $v_2$  (in case the number of agents  $k = 1$ ) or to  $v_1$  (in case  $k \geq 2$ ); from  $v_1$ , depending on history, coalition plays either  $b^\omega$  (when the history is  $v_0v_2v_1$  and hence  $k = 1$ ) or  $a^\omega$  (otherwise) which leads the game back to  $v_0$  (note that at vertex  $v_2$ , choice of actions of the agents is not important, they can collectively play any  $\omega$ -word). However, one can show that there is no memoryless coalition winning strategy. Indeed, the coalition strategy  $a^\omega$  from  $v_1$  is losing for  $k = 1$ , similarly  $b^\omega$  from  $v_1$  is losing for  $k \geq 2$ , and any other strategy is also losing. For instance,  $ba^\omega$  from  $v_0$  is losing because if the game moves to  $v_1$ , coalition has no information on the number of agents and hence any word from  $v_1$  will be losing ( $a^\omega$  is losing for  $k = 1$ ,  $b^\omega$  is losing for  $k \geq 2$ , and similarly for other words).

The rest of the paper is devoted to the proof of the following theorem:

► **Theorem 4.** *The safety coalition problem can be solved in exponential space, and is PSPACE-hard. For positive instances, one can synthesize a winning coalition strategy in exponential space which uses exponential memory; the exponential blowup in the size of the memory is tight.*

### 3 Solving the safety coalition problem

This section is devoted to the proof of Theorem 4. To prove the decidability and establish the complexity upper bound, we construct a tree unfolding of the arena, which is equivalent for deciding the existence of a winning coalition strategy. The unfolding is finite because, if a vertex is repeated along a play, the coalition can play the same  $\omega$ -word as in the first visit, which will be formalized in Section 3.1. We can then show how to solve the safety coalition problem at the tree level in Section 3.2. Synthesis and memory usage are analyzed in Section 3.3, and the running example game is discussed in Section 3.4

The hardness result is shown in Section 3.5 by a reduction from the QBF-SAT problem which is known to be PSPACE-complete [22].

#### 3.1 Finite tree unfolding

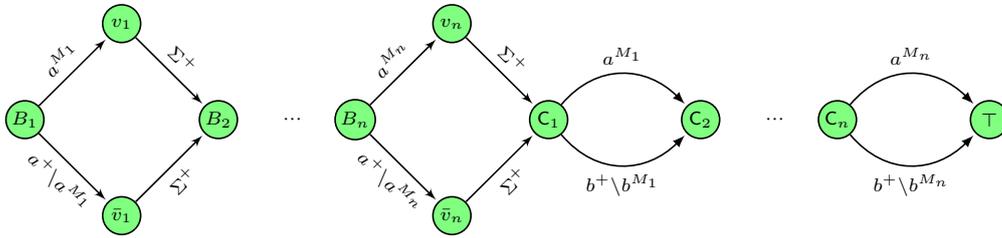
From a parameterized safety game  $\mathcal{G} = (\mathcal{A}, S)$ , we construct a finite tree as follows: we unfold the arena  $\mathcal{A}$  until either some vertex is repeated along a branch or an unsafe vertex is reached. The nodes of the tree are labeled with the corresponding vertices and the edges are labeled with the same regular languages as in the arena  $\mathcal{A}$ . The intuition behind this construction is that if a vertex is repeated in a winning play in  $\mathcal{A}$ , since the winning condition is a safety one, the coalition can play the same strategy as it played in the first occurrence of the vertex. Note however that multiple nodes in the tree may have the same label but different (winning) strategies depending on the history (recall Example 3). This is the reason why we need to consider a tree unfolding abstraction and not a DAG abstraction.

We assume the concept of tree is known. Traditionally, we call a node  $n'$  a *child* of  $n$  (and  $n$  the *parent* of  $n'$ ) if  $n'$  is an immediate successor of  $n$  according to the edge relation; and  $n$  an *ancestor* of  $n'$  if there exists a path from  $n$  to  $n'$  in the tree.

► **Definition 5.** Let  $\mathcal{G} = (\mathcal{A} = \langle V, \Sigma, \Delta \rangle, S)$  be a parameterized safety game and  $v_0 \in V$  an initial vertex. The tree unfolding of  $\mathcal{G}$  is the tree  $\mathcal{T} = \langle N, E, \ell_N, \ell_E \rangle$  rooted at  $n_0 \in N$ , where  $N$  is the finite set of nodes,  $E \subseteq N \times N$  is the set of edges,  $\ell_N : N \rightarrow V$  is the node labeling function,  $\ell_E : N \times N \rightarrow 2^{\Sigma^+}$  is the edge labeling function, and:

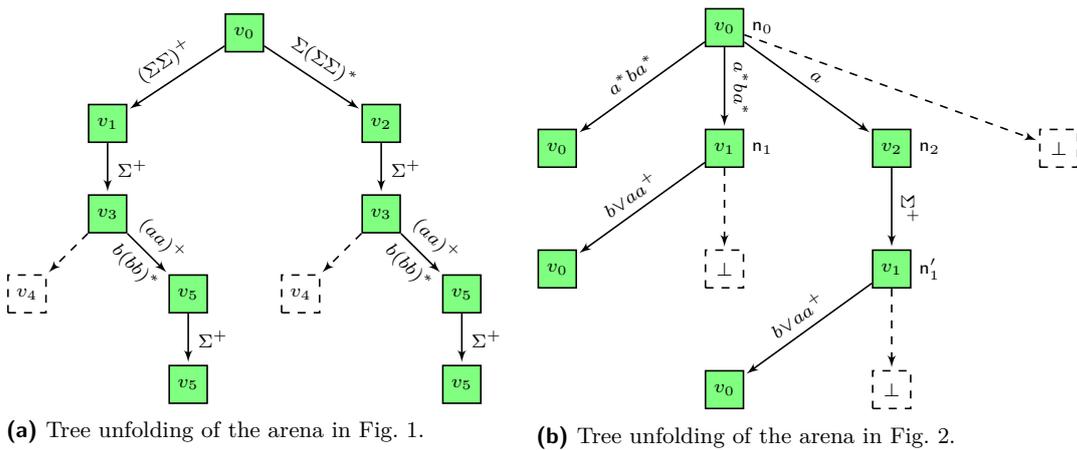
- the root  $n_0$  satisfies  $\ell_N(n_0) = v_0$ ;
- $\forall n \in N$ , if  $\ell_N(n) \in S$  and for every ancestor  $n''$  of  $n$ ,  $\ell_N(n'') \neq \ell_N(n)$ , then  $\forall v' \in V$  such that  $\Delta(v, v')$  is defined, there is  $n'$  a child of  $n$  with  $\ell_N(n') = v'$  and  $\ell_E(n, n') = \Delta(v, v')$ ; otherwise, the node  $n$  has no successor.

Each node in  $\mathcal{T}$  corresponds to a unique history in  $\mathcal{G}$ , and the unfolding is stopped when a vertex repeats or an unsafe vertex is encountered. The set of nodes can be partitioned into  $N = N_{\text{int}} \sqcup N_{\text{leaf}}$  where  $N_{\text{int}}$  is the set of internal nodes and  $N_{\text{leaf}}$  are the leaves of  $\mathcal{T}$  (some leaves are unsafe, some leaves have an equilateral ancestor). By construction, the height of  $\mathcal{T}$  is bounded by  $|V| + 1$  and its branching degree is at most  $|V|$ . The tree unfolding of  $\mathcal{G}$  is hence at most in  $O(|V|^{|V|})$  (and the exponential blowup is unavoidable in general).



■ **Figure 3** Example arena such that the tree unfolding is exponential. All unspecified transitions lead to the sink losing vertex  $\perp$ . Set  $M_i$  denotes multiples of the  $i$ -th prime number. For any play reaching  $C_1$ , for every  $i$ , the number of agents is in  $M_i$  iff the play went through  $v_i$ .

The exponential bound is reached by a family  $(\mathcal{A}_n)_{n \in \mathbb{N}_{>0}}$  of deterministic arenas, shown in Fig. 3, which is an extension of the example in Fig. 1, with  $2n$  many blocks (and,  $O(n)$  many vertices). Observe that to win the game, coalition needs to keep track of the full histories in the first  $n$  blocks, and there are exponentially many such histories; moreover, each such history corresponds to a different node in its unfolding tree.



(a) Tree unfolding of the arena in Fig. 1. (b) Tree unfolding of the arena in Fig. 2.

■ **Figure 4** Tree unfolding examples (green nodes correspond to safe vertices). Notice here that the unsafe leaves (and the edges leading to them) are presented with dashed rectangles (resp. arrows).

► **Example 6.** Fig. 4a and 4b represent the tree unfoldings of the parametrized arenas depicted in Fig. 1 and 2, respectively. On the left picture, the node names are avoided, and in all cases their labels are written within the nodes. The leaf nodes that correspond to unsafe vertices (and the edges leading to them) are presented with dashed rectangles (respectively, arrows). Notice that any leaf node is either labeled with an unsafe vertex (for instance,  $v_4$  in Fig. 4a) or it has a unique ancestor with the same label. These two criteria ensure the tree is always finite (along all branches, some vertex has to repeat within  $|V|$  many steps). However, multiple internal nodes in different branches can have same label but, coalition might have different (winning) strategies depending on their respective histories.

Let  $\mathcal{G} = (\mathcal{A} = \langle V, \Sigma, \Delta \rangle, S)$  be a parameterized safety game with an initial vertex  $v_0$  and  $\mathcal{T} = \langle N, E, \ell_N, \ell_E \rangle$  be the tree unfolding corresponding to  $\mathcal{A}$  with root  $n_0$ . We define the coalition game on  $\mathcal{T}$  as follows.

**History, play and strategy.** Histories in  $\mathcal{T}$  are defined similarly as in  $\mathcal{G}$  (except vertices are replaced by nodes); the set of such histories is denoted  $\text{Hist}_{\mathcal{T}}$ . A *history* in  $\mathcal{T}$  is a finite sequence of nodes  $H = n_0 n_1 \dots n_p \in N^+$  such that for every  $0 \leq j < p$ ,  $(n_j, n_{j+1}) \in E$ . We denote by  $\text{Hist}_{\mathcal{T}}$  the set of all histories in  $\mathcal{T}$ . A *play* in  $\mathcal{T}$  is a maximal history, *i.e.*, a finite sequence of nodes ending with a leaf, thus in  $N_{\text{int}}^+ \cdot N_{\text{leaf}}$ . Note that, contrary to the definition of a play in  $\mathcal{A}$ , a play in  $\mathcal{T}$  is a *finite* sequence of nodes ending in a leaf.

A *coalition strategy* in the unfolding tree is a mapping  $\lambda : N_{\text{int}} \rightarrow \Sigma^\omega$  that assigns to every internal node  $n \in N_{\text{int}}$  an  $\omega$ -word  $\lambda(n)$ . Notice that a coalition strategy in  $\mathcal{T}$  is by definition memoryless (on  $N$ ) which, as we will see later, is sufficient to capture winning strategies of the coalition in  $\mathcal{G}$ . We furthermore extend the definition of node labeling function  $\ell_N$  to a history (resp. play) in the usual way.

Similarly to the parameterized arena setting, we define in a natural way the notions of  $k$ -realizability and of realizability for histories and plays. We also define for a coalition strategy  $\lambda$  in  $\mathcal{T}$  (rooted at  $n_0$ ), and  $k \in \mathbb{N}_{>0}$  the sets  $\text{Out}_{\mathcal{T}}^k(n_0, \lambda)$  and  $\text{Out}_{\mathcal{T}}(n_0, \lambda)$ .

A coalition strategy  $\lambda$  in  $\mathcal{T}$  from  $n_0$  is *winning* for the safety condition defined by the safe set  $S$  if every play in  $\text{Out}_{\mathcal{T}}(n_0, \lambda)$  ends in a leaf with label in  $S$ , *i.e.*, if for every  $R = n_0 \dots n_p \in \text{Out}_{\mathcal{T}}(n_0, \lambda)$ ,  $\ell_N(n_p) \in S$ , written  $\ell_N(\text{Out}_{\mathcal{T}}(n_0, \lambda)) \subseteq S^+$  for short.

**Correctness of the tree unfolding.** Next we show the equivalence of the winning strategies in the safety coalition game, and in the corresponding tree unfolding:

► **Lemma 7.** *Let  $\mathcal{G} = (\mathcal{A} = \langle V, \Sigma, \Delta \rangle, S)$  be a parameterized safety game and  $v_0 \in V$  and  $\mathcal{T} = \langle N, E, \ell_N, \ell_E \rangle$  be the associated tree unfolding with root  $n_0$ . There exists a winning coalition strategy from  $v_0$  in  $\mathcal{G}$  iff there exists a winning coalition strategy from  $n_0$  in  $\mathcal{T}$ .*

**Proof.** Assume first that the coalition of agents has a winning strategy  $\sigma$  in  $\mathcal{G}$ . Any history  $H \in \text{Hist}_{\mathcal{T}}$  can be projected to the history  $\ell_N(H) \in \text{Hist}_{\mathcal{A}}$ . We can hence define for every  $n \in N_{\text{int}}$ ,  $\lambda(n) = \sigma(\ell_N(\iota(n)))$ , where  $\iota$  is the bijection mapping nodes to histories in  $\mathcal{T}$ . To prove that  $\lambda$  is winning in  $\mathcal{T}$ , consider any play  $R = n_0 \dots n_p$  in  $\text{Out}_{\mathcal{T}}(n_0, \lambda)$  and let  $\rho = \ell_N(R) = v_0 \dots v_p$  be its projection in  $\mathcal{G}$ . By construction  $\ell_E(n_i, n_{i+1}) = \Delta(v_i, v_{i+1})$  for each  $i < p$ , and hence from the definition of  $\lambda$ ,  $\rho$  is a history in  $\mathcal{G}$  induced by  $\sigma$ . Since  $\sigma$  is winning,  $\rho$  only visits *safe* vertices. In particular,  $\ell_N(n_p) \in S$ . Since this is true for every play induced by  $\lambda$ , strategy  $\lambda$  is winning from  $n_0$  in  $\mathcal{T}$ .

For the other direction, assume that  $\lambda$  is a winning coalition strategy from  $n_0$  in  $\mathcal{T}$ . The tree will be the basis of a memory structure sufficient to win the game; we thus explain how histories in  $\mathcal{G}$  can be mapped to nodes of  $\mathcal{T}$ . We first define a mapping  $\text{zip} : \text{Hist}_{\mathcal{A}} \rightarrow \text{Hist}_{\mathcal{T}}$

that summarizes any history in  $\mathcal{A}$  to its *virtual history* where each vertex appears at most once. Intuitively,  $\text{zip}$  greedily shortens a history by appropriately removing the loops until an unsafe vertex is encountered (if any). The mapping  $\text{zip}$  is defined inductively, starting with  $\text{zip}(v_0) = v_0$ , and letting for every  $h \in \text{Hist}_{\mathcal{A}}$  and every  $v' \in V$  such that  $h \cdot v' \in \text{Hist}_{\mathcal{A}}$ ,

$$\text{zip}(h \cdot v') = \begin{cases} \text{zip}(h) \cdot v' & \text{if } v' \text{ does not appear in } \text{zip}(h) \\ v_0 \dots v' \sqsubseteq \text{zip}(h) & \text{otherwise} \end{cases}$$

The mapping  $\text{zip}$  is well-defined (by construction, for every history  $h$ , any vertex appears at most once in  $\text{zip}(h)$ , so that when  $v'$  appears in  $\text{zip}(h)$ , there is a unique prefix of  $\text{zip}(h)$  ending with  $v'$ ). Note that, since unsafe vertices are sinks, as soon as  $h$  reaches an unsafe vertex, the value of  $\text{zip}(h)$  stays unchanged.

► **Lemma 8.** *The application  $\beta: n \mapsto \ell_N(\iota(n))$  defines a bijection between  $N_{\text{int}} \cup \{n \in N_{\text{leaf}} \mid \ell_N(n) \notin S\}$  and the set  $Z = \{\text{zip}(h) \mid h \in \text{Hist}_{\mathcal{A}}\}$ .*

**Proof.** It is first obvious that this application is injective, since two nodes of the tree corresponds to different histories in  $\mathcal{A}$  which all belong to  $Z$ .

This application is surjective: pick  $h \in Z$ ; then,  $h$  has no repetition; furthermore it forms a real history in  $\mathcal{G}$ , which implies that it can be read as the label of some history in the tree unfolding. ◀

We write  $\alpha = \beta^{-1}$ . Using the  $\text{zip}$  function and  $\alpha$ , from a coalition strategy  $\lambda$  in  $\mathcal{T}$ , we define a coalition strategy  $\sigma$  in  $\mathcal{G}$  by applying  $\sigma$  to the virtual histories: for every history  $h = v_0 \dots v_p$  in  $\mathcal{G}$  we let  $\sigma(h) = \lambda(\alpha(\text{zip}(h)))$  whenever  $\alpha(\text{zip}(h)) \in N_{\text{int}}$  and  $\sigma(h)$  is set arbitrarily otherwise (recall that if  $\alpha(\text{zip}(h))$  is a leaf node, then  $h$  is actually already a losing history).

Towards a contradiction, assume that  $\sigma$  is not winning in  $\mathcal{G}$ . Consider, some number of agents  $k \in \mathbb{N}_{>0}$ , and a losing play with  $k$  agents:  $\rho = v_0 v_1 \dots \in \text{Out}_{\mathcal{A}}^k(v_0, \sigma)$ . Let  $h' = v_0 v_1 \dots v_q \sqsubseteq \rho$  be the shortest prefix of  $\rho$  ending in an unsafe vertex  $v_q \notin S$ , and write  $\text{zip}(h') = v_0 v_{i_1} \dots v_q$  for the corresponding virtual history. By definition of  $\sigma$ ,  $\text{zip}(h')$  is a  $k$ -outcome of  $\sigma$  from  $v_0$ . Moreover, the corresponding play  $R = \iota(\alpha(\text{zip}(h'))) = n_0 n_{i_1} \dots n_q$  in  $\mathcal{T}$ , belongs to the  $k$ -outcome of  $\lambda$  from  $n_0$ . Since  $v_q \notin S$ ,  $\lambda$  is not winning in  $\mathcal{T}$ ; which is a contradiction. We conclude that  $\sigma$  is a winning coalition strategy in  $\mathcal{G}$ . ◀

► **Example 9.** We illustrate the  $\text{zip}$  function on the arena in Fig. 2. Take  $h = v_0 v_1 v_0 v_1$ . First,  $\text{zip}(v_0) = v_0$ ; then  $\text{zip}(v_0 v_1) = \text{zip}(v_0) \cdot v_1 = v_0 v_1$ ;  $\text{zip}(v_0 v_1 v_0) = v_0$  (which is the unique prefix of  $\text{zip}(v_0 v_1) = v_0 v_1$ , ending at  $v_0$ ); finally  $\text{zip}(v_0 v_1 v_0 v_1) = \text{zip}(v_0 v_1 v_0) \cdot v_1 = v_0 v_1$ . Then the function  $\alpha$  uniquely maps each virtual history (*i.e.*,  $\text{zip}(h)$ ) ending at a safe vertex to an internal node in the tree, which is the heart of the proof of Lemma 7.

### 3.2 Existence of winning coalition strategy on the tree unfolding

In the previous subsection, we showed that the safety coalition problem reduces to solving the existence of a winning coalition strategy in the associated finite tree unfolding.

To solve the latter, from the tree unfolding  $\mathcal{T}$ , we construct a deterministic (safety) automaton over the alphabet  $\Sigma^m$ , where  $m = |N_{\text{int}}|$ , which accepts the  $\omega$ -words corresponding to winning coalition strategies in  $\mathcal{T}$ . More precisely, since  $(\Sigma^m)^\omega$  and  $(\Sigma^\omega)^m$ , understood as the set of  $m$ -tuples of  $\omega$ -words over  $\Sigma$ , are in one-to-one correspondence, an infinite word  $\mathbf{w} \in (\Sigma^m)^\omega$  corresponds to  $m$  infinite words  $\mathbf{w}_n$ , one for each internal node  $n \in N_{\text{int}}$ , thus representing a coalition strategy in  $\mathcal{T}$ .

## 39:10 Synthesizing Safe Coalition Strategies

Fix  $\mathcal{G} = (\mathcal{A}, S)$  a parameterized safety game with  $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$  and  $v_0 \in V$  an initial vertex. We assume for every  $(v, v') \in V \times V$  such that  $\Delta(v, v') \neq \emptyset$ ,  $\Delta(v, v')$  is given as a complete DFA over  $\Sigma$ . Those will be given as inputs to the algorithm.

Let  $\mathcal{T} = \langle N, E, \ell_N, \ell_E \rangle$  be the associated unfolding tree with root  $n_0$ . For the rest of this section, we fix an arbitrary ordering on the internal nodes of  $\mathcal{T}$  and on the edges:  $N_{\text{int}} = \{n_1, \dots, n_m\}$  and  $E = \{e_1, \dots, e_r\}$ , with  $|N_{\text{int}}| = m$  and  $|E| = r$ .

Assuming there are  $t$  leaves –thus  $t$  plays– in  $\mathcal{T}$ , for every  $1 \leq i \leq t$ , the  $i$ -th play is denoted  $n_0^i \dots n_{z_i}^i$  with  $n_0^i = n_0$ ,  $\forall j < z_i$ ,  $n_j^i \in N_{\text{int}}$  and  $n_{z_i}^i \in N_{\text{leaf}}$ . Also, for  $0 \leq j < z_i$ , we note  $e_j^i = (n_j^i, n_{j+1}^i)$ .

The automaton for the winning coalition strategies in  $\mathcal{T}$  builds on the finite automata that recognize the regular languages that label edges of  $\mathcal{T}$ . For each edge  $e \in E$ , let us write  $\mathcal{B}_e = (Q_e, \Sigma, \delta_e, q_e^0, F_e)$  for the complete DFA over  $\Sigma$  such that  $L(\mathcal{B}_e) = \ell_E(e)$ . (Here  $Q_e$  is the set of states,  $\delta_e$  the transition function,  $q_e^0 \in Q_e$  the initial state and  $F_e$  the set of accepting states.) Note that some of the  $\mathcal{B}_e$ 's are identical since they correspond to the same original edge of  $\mathcal{G}$ .

We then define a deterministic safety automaton  $\mathcal{B} = (Q, \Sigma^m, \delta, q^0, F)$  that simulates all  $\mathcal{B}_e$ 's in parallel and accepts  $\omega$ -words over alphabet  $\Sigma^m$  if every prefix satisfies the following: on every branch of the tree, if all corresponding  $\mathcal{B}_e$ 's accept, then the leaf is labeled by a safe vertex. Formally,  $Q \subseteq Q_1 \times \dots \times Q_r$  is the set of states;  $q^0 = (q_1^0, \dots, q_r^0)$  is the initial state; the transition relation  $\delta$  executes the  $r$  automata  $\mathcal{B}_e$ 's componentwise: if letter  $u \in \Sigma^m$  is read, then make the  $s$ -th component mimic  $\mathcal{B}_{e_s}$  by reading the  $l$ -th letter of  $u$ , where  $l$  is the index (in the enumeration fixed above) of the source node of  $e_s$ ; and the accepting set  $F$  is composed of all states  $q = (q_1, \dots, q_r)$  that satisfy the following Boolean formula:

$$\varphi = \bigwedge_{1 \leq i \leq t} \varphi_i \quad \text{where } \varphi_i = \left( \left[ \bigwedge_{0 \leq j < z_i} q_{e_j^i} \in F_{e_j^i} \right] \Rightarrow \ell_N(n_{z_i}^i) \in S \right).$$

Note that  $\mathcal{B}$  is equipped with a safety acceptance condition:<sup>2</sup> an infinite run  $\zeta = q^0 q^1 q^2 \dots$  of  $\mathcal{B}$  is accepting if for every  $k \geq 1$ ,  $q^k \in F$ , and  $L(\mathcal{B})$  consists of all words  $\mathbf{w}$  whose unique corresponding run is accepting.

Intuitively,  $\varphi_i$  expresses that if for some number of agents  $k$ , the languages along the  $i$ -th maximal path contain the  $k$ -length prefixes of corresponding  $\omega$ -words (which means the induced play is  $k$ -realizable), then it should lead to a safe leaf; and then  $\varphi$  ensures that this should be true for all plays. This is formalized in the next lemma.

► **Lemma 10.** *Let  $\lambda : N_{\text{int}} \rightarrow \Sigma^\omega$  be a coalition strategy in  $\mathcal{T}$ . Then,  $\lambda$  is winning if and only if  $(\lambda(n_1), \lambda(n_2), \dots, \lambda(n_m)) \in L(\mathcal{B})$ .*

Notice that in the above statement, we slightly abuse notation:  $(\lambda(n_1), \lambda(n_2), \dots, \lambda(n_m))$  belongs to  $(\Sigma^\omega)^m$ , however it uniquely maps to a word in  $(\Sigma^m)^\omega$ , that can thus be read in  $\mathcal{B}$ .

**Proof.** Assume  $\lambda : N_{\text{int}} \rightarrow \Sigma^\omega$  is a winning coalition strategy in  $\mathcal{T}$ , and consider the corresponding word  $\mathbf{w} = (\lambda(n_1), \lambda(n_2), \dots, \lambda(n_m))$ . Let us show that  $\mathbf{w} \in L(\mathcal{B})$ . Consider the infinite run  $\zeta = q^0 q^1 \dots$  of  $\mathcal{B}$  on  $\mathbf{w}$ . Fix a number of agents  $k \in \mathbb{N}_{>0}$ . Since  $\lambda$  is winning, any  $k$ -length prefix of  $\lambda$ -induced play in  $\text{Out}_{\mathcal{T}}^k(n_0, \lambda)$  is winning. Therefore for any  $1 \leq i \leq t$

<sup>2</sup> This is a slight abuse of language since  $q^0$  need not be in the safe set  $F$ .

such that  $n_0^i \dots n_{z_i}^i$  is in  $\text{Out}_{\mathcal{T}}^k(n_0, \lambda)$ , the play satisfies for all  $0 \leq j < z_i$ ,  $[\lambda(n_j^i)]_{\leq k} \in \ell_E(e_j^i)$  and furthermore,  $\ell_N(n_{z_i}^i) \in S$ ; and hence  $q^k \models \varphi_i$ . Otherwise, if a length  $k$ -play is not induced by  $\lambda$ , then  $\varphi_i$  is vacuously true for that  $i \leq t$ . We conclude  $q^k \models \varphi$ . Since this is true for every  $k \in \mathbb{N}_{>0}$ ,  $\mathbf{w} \in L(\mathcal{B})$ .

Let now  $\lambda$  be an arbitrary coalition strategy, and assume  $\mathbf{w} = (\lambda(n_1), \lambda(n_2), \dots, \lambda(n_m)) \in L(\mathcal{B})$  with  $\zeta = q^0 q^1 \dots$  the accepting run on  $\mathbf{w}$ . Then for any number of agents  $k \in \mathbb{N}_{>0}$ ,  $q^k \in F$ , and hence  $q^k \models \varphi$ . Therefore for all  $1 \leq i \leq t$ ,  $q^k \models \varphi_i$ . Fix any such  $i$ ; let  $n_0^i \dots n_{z_i}^i$  be the  $i$ -th maximal path, and write  $q^k = (q_1^k, \dots, q_r^k)$ . Then for some  $0 \leq j < z_i$ , the condition  $q_{e_j^i}^k \in F_{e_j^i}$  implies  $[\lambda(n_j^i)]_{\leq k} \in \ell_E(e_j^i)$ . In case the above is true for all  $0 \leq j < z_i$ , we conclude  $n_0^i \dots n_{z_i}^i \in \text{Out}_{\mathcal{T}}^k(n_0, \lambda)$  and  $\varphi_i$  ensures that  $\ell_N(n_{z_i}^i) \in S$ . Otherwise,  $n_0^i \dots n_{z_i}^i \notin \text{Out}_{\mathcal{T}}^k(n_0, \lambda)$ . Finally  $\varphi$  ensures all  $k$ -length prefixes of  $\lambda$ -induced plays in  $\mathcal{T}$  are winning. Since this is true for any number of agents  $k$ ,  $\lambda$  is a winning coalition strategy in  $\mathcal{T}$ .  $\blacktriangleleft$

We now have all ingredients to solve the safety coalition problem, and to state a complexity upper-bound. As mentioned earlier, we assume that the arena is initially given with all associated complete DFAs (used by all  $\mathcal{B}_e$ ) in the input.

► **Theorem 11.** *The safety coalition problem is in EXPSPACE.*

**Proof.** Solving the safety coalition problem reduces to checking non-emptiness of the language recognized by the deterministic safety automaton  $\mathcal{B}$ . We adapt to our setting the standard algorithm which runs in non-deterministic logarithmic space, when  $\mathcal{B}$  is given as an input.

We write  $N$  for the number of states of  $\mathcal{B}$  and notice that  $N$  is doubly exponential in  $|V|$ , the number of vertices of the initial arena  $\mathcal{A}$  (each state of  $\mathcal{B}$  is an exponential-size vector of states of automata given in the input). We do not build  $\mathcal{B}$  a priori. Instead, we non-deterministically guess a safe prefix of length at most  $N$  (we only keep written two consecutive configurations and keep a counter to count up to  $N$ ), and then a safe lasso on the last state of length at most  $N$ .

Provided one can check “easily” whether a state of  $\mathcal{B}$  is safe, the described procedure runs in non-deterministic exponential space, hence can be turned into a deterministic exponential space algorithm, by Savitch’s theorem.

It remains to explain how one checks that a given state in  $\mathcal{B}$  is safe. Formula  $\varphi$  is a SAT formula exponential in the size of  $\mathcal{A}$ , which can therefore be solved in exponential space as well.

Overall, we conclude that the safety coalition problem is in EXPSPACE.  $\blacktriangleleft$

### 3.3 Synthesizing a winning coalition strategy

We assume all the notations of the two previous subsections, and we explain how we build a winning coalition strategy. From an accepting word of the form  $\mathbf{u} \cdot \mathbf{v}^\omega$  in  $\mathcal{B}$  (where  $\mathbf{u} \in (\Sigma^m)^*$  and  $\mathbf{v} \in (\Sigma^m)^+$ ), one can synthesize a winning strategy  $\lambda$  in  $\mathcal{T}$  by:

$$\lambda(n_i) = \mathbf{u}_i \cdot \mathbf{v}_i^\omega \quad \text{for every } n_i \in N_{\text{int}}.$$

Then it is easy to transfer to a winning coalition strategy  $\sigma$  in  $\mathcal{G}$  by defining

$$\sigma(h) = \lambda(\alpha(\text{zip}(h))) \quad \text{for every history } h \in \text{Hist}_{\mathcal{A}},$$

that is, the  $\omega$ -word corresponding to the internal node representing its virtual history. Recall that, following the proof of Lemma 7,  $\text{zip}$  assigns to every history its virtual history (by greedily removing all the loops) and  $\alpha$  associates to a virtual history its corresponding node in the tree  $\mathcal{T}$ .

## 39:12 Synthesizing Safe Coalition Strategies

► **Proposition 12.** *If there is a winning coalition strategy for a game  $\mathcal{G} = (\mathcal{A}, S)$ , then there is one which uses exponential memory, which can be computed in exponential space. Furthermore, winning might indeed require exponential memory.*

**Proof.** The tree unfolding can be seen as a memory structure for a winning strategy. Indeed, consider the memory set defined by  $N_{\text{int}}$ , starting from memory state  $n_0$ . Define the application  $\text{upd} : N_{\text{int}} \times V \rightarrow N_{\text{int}}$  by  $\text{upd}(n, v') = n'$  such that  $v' \in S$  whenever

- either  $n' \in N_{\text{int}}$  is a child of  $n$  such that  $\ell_N(n') = v'$
- or  $n' \in N_{\text{int}}$  is an ancestor of  $n'' \in N_{\text{leaf}}$  such that  $\ell_N(n'') = \ell_N(n') = v'$ , and  $n''$  is a child of  $n$ .

We also define the application  $\text{act} : N_{\text{int}} \times V \rightarrow \Sigma^\omega$  by  $\text{act}(n, v) = \lambda(n)$ .

Then, it is easy to see that winning strategy  $\sigma$  can be defined using memory  $N_{\text{int}}$  and applications  $\text{upd}$  and  $\text{act}$ .

Furthermore, though the  $\omega$ -words extracted from  $\mathcal{B}$  can be of doubly-exponential size, their computation and the overall procedure only requires exponential space.

For the lower bound, we show the following lemma.

► **Lemma 13.** *There is a family of games  $(\mathcal{G}_n)_n$  such that the size of  $\mathcal{G}_n$  is polynomial in  $n$  but winning coalition strategies require exponential memory.*

**Proof.** We again consider the game of Fig. 3, whose description can be made in polynomial time (since the  $i$ -th prime number uses only  $\log(i)$  bits in its binary representation). We have already seen that its tree unfolding has exponential size. We will argue why exponential memory is required, that is, one cannot do better than the tree memory structure.

First notice that there is a winning coalition strategy: play  $a^\omega$  at every vertex  $B_i$ , and  $a^\omega$  (resp.  $b^\omega$ ) at vertex  $C_i$  if the history went through  $v_i$  (resp.  $\bar{v}_i$ ). This strategy can be implemented using the memory given by the tree unfolding.

Assume one can do better and have a memory structure of size strictly smaller than  $2^n$ . Then, arriving in vertex  $C_1$ , there are at least two different histories leading to the same memory state, hence the coalition strategy will select exactly the same  $\omega$ -words in all vertices  $C_1, C_2, \dots, C_n$ . We realize that it cannot be winning since the two histories disagree at least on a predicate “be a multiple of the  $i$ -th prime number”. Contradiction. ◀

◀

### 3.4 Illustration of the construction

We illustrate the construction on one example.

► **Example 14.** Fig. 6 represents part of the automaton  $\mathcal{B}$  corresponding to the tree  $\mathcal{T}$  in Fig. 4b (that is, the tree unfolding of the arena in Fig. 2). The automata  $\mathcal{B}_e$  for the languages labeling the edges of  $\mathcal{T}$  are depicted in Fig. 5. Here notice that each state of  $\mathcal{B}$  has as many components as the number of edges leading to a safe node in  $\mathcal{T}$ , we did not consider the edges leading to  $\perp$ . This is without loss of any generality: the language on any “unsafe” edge leading to  $\perp$ , in this example, are disjoint from the languages on the edges leading to its *siblings* (other children of its parent node). The first three positions in a state of  $\mathcal{B}$ , presented as a single cell in the picture, correspond to the outgoing edges of the *root*  $n_0$  of  $\mathcal{T}$  (hence they follow the same component in  $\Sigma^m$ ), and the other positions correspond to the other edges (in some chosen order). “ $\times$ ” in a component of a state denotes the non-accepting sink state of the corresponding automaton (as mentioned in Fig. 5). Finally, here we have only shown the accepting states (marked in blue) and some of the non-accepting states. Indeed

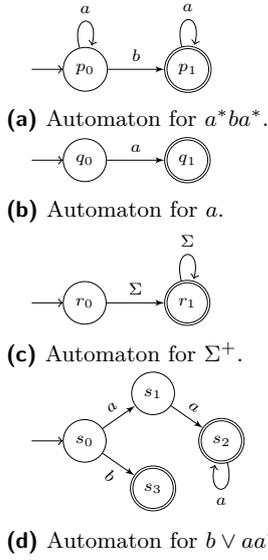


Figure 5 Automata corresponding to the input languages of Fig. 2. The automata are not complete for sake of readability; all unspecified letters lead to a (sink) non-accepting state “×”.

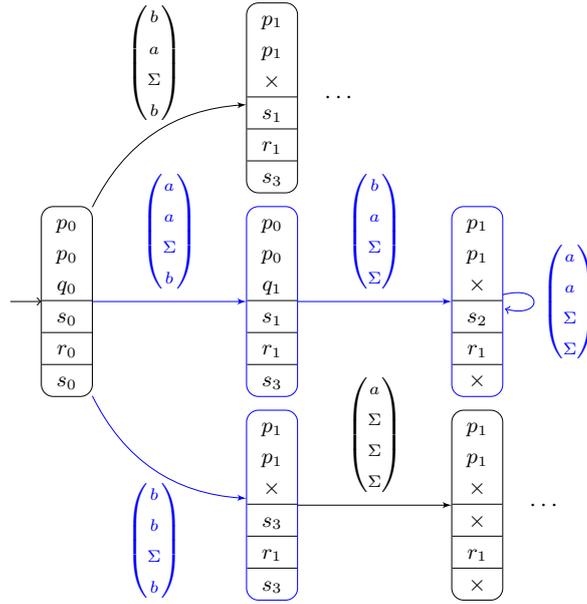


Figure 6 Automaton  $\mathcal{B}$  corresponding to the tree given in Fig. 4b. Here we have only shown the accepting states (marked in blue) and some of the non-accepting states. Further explanations are given in Example 14.

one can verify that the states which are colored in blue satisfy the formula  $\varphi$ ; for instance, the state  $(p_1, p_1, \times, s_2, r_1, \times)$  on the right corresponds to the two maximal paths  $v_0v_0$  and  $v_0v_1v_0$  in  $\mathcal{T}$  (notice we used the node labels here), and all of them lead to safe nodes. The infinite execution in blue (*i.e.*, all the words in  $(a, a, \Sigma, b) \cdot (b, a, \Sigma, \Sigma) \cdot (a, a, \Sigma, \Sigma)^\omega$ ) corresponds to the winning coalition strategies in the tree: for instance,  $\lambda(n_0) = aba^\omega$ ;  $\lambda(n_1) = a^\omega$ ; for any  $a \in \Sigma$ ,  $\lambda(n_2) = a^\omega$ ; and  $\lambda(n'_1) = b^\omega$  is a winning coalition strategy (note here, for instance, that at node  $n_2$ , any word from  $\Sigma^\omega$  could be played).

### 3.5 PSPACE lower bound

We show the safety coalition problem is PSPACE-hard by reduction from QBF-SAT, which is known to be PSPACE-complete [22]. The construction is inspired by the one in [4], where the first agent was playing against the coalition of all other agents, with a reachability objective.

► **Proposition 15.** *The safety coalition problem is PSPACE-hard.*

**Proof sketch.** Let  $\varphi = \exists x_1 \forall x_2 \exists x_3 \dots \forall x_{2r} \cdot (C_1 \wedge C_2 \wedge \dots \wedge C_m)$  be a quantified Boolean formula in prenex normal form, where for every  $1 \leq h \leq m$ ,  $C_h = \ell_{h,1} \vee \ell_{h,2} \vee \ell_{h,3}$ , and for every  $1 \leq j \leq 3$ ,  $\ell_{h,j} \in \{x_i, \neg x_i \mid 1 \leq i \leq 2r\}$  are the literals.

In the reduction, we use sets of natural numbers (that represent the number of agents) corresponding to multiples of primes. Let thus  $p_i$  be the  $i$ -th prime number and  $M_i$  the set of all non-zero natural numbers that are multiples of  $p_i$ . For simplicity, we write  $a^{M_i}$  to denote the set of words in  $(a^{p_i})^+$ , that is words from  $a^+$  whose length is a multiple of  $p_i$ . It is well-known that the  $i$ -th prime number requires  $O(\log(i))$  bits in its binary representation, hence the description of each of the above languages is polynomial in the size of  $\varphi$ .

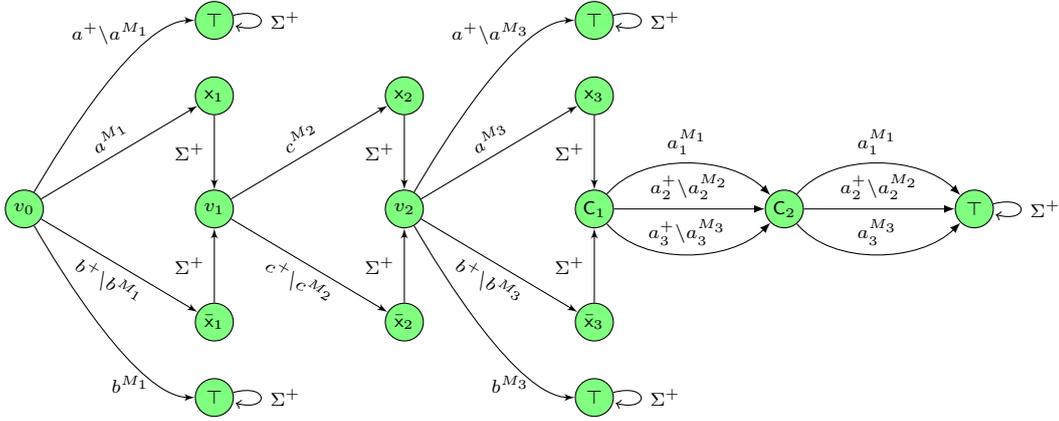
### 39:14 Synthesizing Safe Coalition Strategies

From  $\varphi$ , we construct an arena  $\mathcal{A}_\varphi = \langle V, \Sigma, \Delta \rangle$  as follows:

- $V = \{v_0, v_1, \dots, v_{2r-1}, v_{2r}\} \cup \{x_1, \bar{x}_1, \dots, x_{2r}, \bar{x}_{2r}\} \cup \{C_1, C_2, \dots, C_m, C_{m+1}\} \cup \{\perp, \top\}$ , where we identify some vertices:  $v_{2r} = C_1$ , and  $C_{m+1} = \top$ .
- $\Sigma = \{a, b, c\} \cup \bigcup_{1 \leq i \leq 2r} \{a_i\}$ .
- For every  $0 \leq s \leq r-1$ , every  $1 \leq i \leq 2r$  and every  $1 \leq h \leq m$ :
  1.  $\Delta(v_{2s}, x_{2s+1}) = a^{M_{2s+1}}$  and  $\Delta(v_{2s}, \bar{x}_{2s+1}) = b^+ \setminus b^{M_{2s+1}}$
  2.  $\Delta(v_{2s}, \top) = (a^+ \setminus a^{M_{2s+1}}) \cup b^{M_{2s+1}}$
  3.  $\Delta(v_{2s+1}, x_{2s+2}) = c^{M_{2s+2}}$  and  $\Delta(v_{2s+1}, \bar{x}_{2s+2}) = c^+ \setminus c^{M_{2s+2}}$
  4.  $\Delta(x_i, v_i) = \Sigma^+$  and  $\Delta(\bar{x}_i, v_i) = \Sigma^+$
  5.  $\Delta(C_h, C_{h+1}) = \bigcup_{1 \leq j \leq 3} L_{h,j}$  where  $L_{h,j} = a_i^{M_i}$  if  $\ell_{h,j} = x_i$ ;  $L_{h,j} = a_i^+ \setminus a_i^{M_i}$  if  $\ell_{h,j} = \bar{x}_i$ .

To obtain a complete arena, all unspecified transitions lead to vertex  $\perp$ .

On the arena  $\mathcal{A}_\varphi$ , we consider the safety coalition game  $\mathcal{G}_\varphi = (\mathcal{A}_\varphi, S)$  with  $S = V \setminus \{\perp\}$ . The construction is illustrated on a simple example with 3 variables and 2 clauses in Fig. 7.



■ **Figure 7** Parameterized arena for the formula  $\varphi = \exists x_1 \forall x_2 \exists x_3 \cdot (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$ . All unspecified transitions lead to the sink losing vertex  $\perp$ . Set  $M_i$  denotes multiples of the  $i$ -th prime number. Vertex  $x_i$  (resp.  $\bar{x}_i$ ) represents setting variable  $x_i$  to **true** (resp. **false**). For any play reaching  $C_1$ , for every  $i$ , the number of agents is in  $M_i$  iff the play went through  $x_i$ .

From  $v_0$ , a first phase up to  $v_{2r} = C_1$  consists in choosing a valuation for the variables. The coalition chooses the truth values of existentially quantified variables  $x_{2s+1}$  in vertices  $v_{2s}$ : it plays  $a^\omega$  for **true**, and  $b^\omega$  for **false**. In the first (resp. second) case, if the number of agents involved in the coalition is (resp. is not) a multiple of  $p_{2s+1}$ , then the game proceeds to the next variable choice, otherwise the safe  $\top$  state is reached (forever).

For universally quantified variables the coalition must play  $c^\omega$  in vertices  $v_{2s+1}$ , as any other choice would immediately lead to the sink losing vertex  $\perp$ ; the choice of the assignment then only depends on whether the number of agents involved in the coalition is a multiple of  $p_{2s+2}$  (in which case variable  $x_{2s+2}$  is assigned **true**) or not (in which case variable  $x_{2s+2}$  is assigned **false**).

Hence, depending on the number of agents involved in the coalition, either the play will proceed to state  $v_{2r} = C_1$ , in which case the number of agents characterizes the valuation of the variables (it is a multiple of  $p_i$  if and only if variable  $x_i$  is set to **true**); or it will have escaped to the safe state  $\top$ .

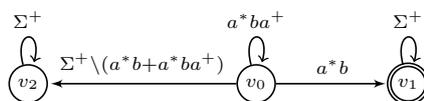
Note that in terms of information, the coalition learns progressively assignments (thanks to the visit to either vertex  $x_i$  or vertex  $\bar{x}_i$ ). Note also that the coalition can never learn assignments of next variables in advance (it can only know whether it is a multiple of previously seen prime numbers, hence of previously quantified variables, not of variables quantified afterwards).

From  $C_1$ , a second phase starts where one checks whether the generated valuation makes all clauses in  $\varphi$  true. If it is the case, sequentially, the coalition chooses for every clause a literal that makes the clause true. The arena forces these choices to be consistent with the valuation generated in the first phase. For instance, on the example of Fig. 7, to set  $x_1$  to **true** in the first phase, the coalition must play  $a^\omega$ , and only plays with a number of agents in  $M_1$  do not move to  $\top$  and continue the first phase from  $x_1$ . Then, in the second phase, for instance for the first clause, one can choose literal  $\ell_{1,1} = x_1$  by playing  $a_1^\omega$ . The same language  $-a_1^{M_1}$  labels the edge from  $C_1$  to  $C_2$ , so that the play proceeds to  $C_2$ . More generally, if  $a_i^\omega$  leads from  $C_h$  to  $C_{h+1}$  with number of agents in  $M_i$ , this means that  $x_i$  was visited, hence indeed  $x_i$  was set to **true**. On the contrary, if  $a_i^\omega$  leads from  $C_h$  to  $C_{h+1}$  with number of agents not in  $M_i$ , this means that  $\bar{x}_i$  was visited, hence indeed  $x_i$  was set to **false**.

The above reduction ensures the following equivalence: there is a winning coalition strategy in the game  $\mathcal{G}_\varphi = (\mathcal{A}_\varphi, S)$  if and only if  $\varphi$  is true. ◀

The proof in full details can be found in the arxiv version of this paper [5].

## 4 Future work



■ **Figure 8** Example of a reachability coalition game: a winning coalition strategy is that Agent  $n$  plays  $a$  for the first  $n-1$  rounds, then  $b$  for one round, and finally  $a$  forever.

In this paper, we focused on and obtained results for the coalition problem for safety objectives. The problem can obviously be defined for other objectives. The finite tree unfolding technique will not be correct in a general setting. We illustrate this on the game arena in Fig. 8. In this example, the goal is to collectively reach the target  $v_1$ . One can do so if, at  $v_0$ , the last agent involved plays a  $b$  whereas all the others play  $a$ . On the other hand, at  $v_0$ , it is safe if exactly one agent plays a  $b$  and the other plays an  $a$ . Coalition has a winning strategy: Agent  $n$  plays action  $a$  for the first  $n-1$  rounds, then plays  $b$ , and finally plays  $a$  for the remaining steps. Doing so, each agent will in turn play action  $b$ , and when the last agent does so, the play will reach  $v_1$ . Notice that, for each agent (and hence for the coalition), the strategy when going through  $v_0$  differs at some step (at every step from the coalition point-of-view), so no finite tree unfolding will be correct.

As future work, we obviously would like to match lower and upper bounds for the safety coalition problem, but more importantly we would like to investigate more general objectives.

---

## References

- 1 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 564–575. IEEE Computer Society Press, 1998. doi:10.1109/SFCS.1998.743507.

- 2 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002. doi:10.1145/585265.585270.
- 3 Krzysztof Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, May 1986. doi:10.1016/0020-0190(86)90071-2.
- 4 Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Concurrent parameterized games. In *Proceedings of the 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'19)*, volume 150 of *LIPICs*, pages 31:1–31:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.FSTTCS.2019.31.
- 5 Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Synthesizing safe coalition strategies, 2020. URL: <https://arxiv.org/abs/2008.03770>.
- 6 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Logical Methods in Computer Science*, 15(3), 2019. doi:10.23638/LMCS-15(3:6)2019.
- 7 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'14)*, volume 8412 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2014. doi:10.1007/978-3-642-54830-7\_9.
- 8 Dietmar Berwanger, Lukasz Kaiser, and Bernd Puchala. A perfect-information construction for coordination in games. In *Proceedings of the 31th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, volume 13 of *LIPICs*, pages 387–398. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICs.FSTTCS.2011.387.
- 9 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.
- 10 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent games. *Logical Methods in Computer Science*, 11(2:9), 2015. doi:10.2168/LMCS-11(2:9)2015.
- 11 Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. Reachability and coverage planning for connected agents. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 144–150. ijcai.org, 2019. doi:10.24963/ijcai.2019/21.
- 12 Thomas Colcombet, Nathanaël Fijalkow, and Pierre Ohlmann. Controlling a random population. In *Proceedings of the 23rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'20)*, volume 12077 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2020. doi:10.1007/978-3-030-45231-5\_7.
- 13 Giorgio Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods in System Design*, 23(3):257–301, 2003. doi:10.1023/A:1026276129010.
- 14 E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *Proceedings of the 17th International Conference on Automated Deduction (CADE'00)*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000. doi:10.1007/10721959\_19.
- 15 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPICs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.1.
- 16 Dana Fisman, Orna Kupferman, and Yoav Lustig. Rational synthesis. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2010. doi:10.1007/978-3-642-12002-2\_16.

- 17 Igor Konnov, Helmut Veith, and Josef Widder. What you always wanted to know about model checking of fault-tolerant distributed algorithms. In *Proceedings of the 10th International Andrei Ershov Informatics Conference (PSI'15)*, volume 9609 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2015. doi:10.1007/978-3-319-41579-6\_2.
- 18 Corto Mascle, Mahsa Shirmohammadi, and Patrick Totzke. Controlling a random population is EXPTIME-hard, 2019. URL: <https://arxiv.org/abs/1909.06420>.
- 19 Swarup Mohalik and Igor Walukiewicz. Distributed games. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2003. doi:10.1007/978-3-540-24597-1\_29.
- 20 Gary L. Peterson and John H. Reif. Multiple-person alternation. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science (FOCS'79)*, pages 348–363. IEEE Computer Society Press, 1979. doi:10.1109/SFCS.1979.25.
- 21 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'90)*, volume 2, pages 746–757. IEEE Computer Society Press, 1990. doi:10.1109/FSCS.1990.89597.
- 22 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.
- 23 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2011. doi:10.1007/978-3-642-23217-6\_32.



# Dynamic Network Congestion Games

**Nathalie Bertrand** 

Univ Rennes, Inria, CNRS, IRISA, Rennes, France  
nathalie.bertrand@inria.fr

**Nicolas Markey** 

Univ Rennes, Inria, CNRS, IRISA, Rennes, France  
nicolas.markey@irisa.fr

**Suman Sadhukhan**

Univ Rennes, Inria, CNRS, IRISA, Rennes, France  
suman.sadhukhan@inria.fr

**Ocan Sankur** 

Univ Rennes, Inria, CNRS, IRISA, Rennes, France  
ocan.sankur@irisa.fr

---

## Abstract

---

Congestion games are a classical type of games studied in game theory, in which  $n$  players choose a resource, and their individual cost increases with the number of other players choosing the same resource. In network congestion games (NCGs), the resources correspond to simple paths in a graph, *e.g.* representing routing options from a source to a target. In this paper, we introduce a variant of NCGs, referred to as *dynamic NCGs*: in this setting, players take transitions synchronously, they select their next transitions dynamically, and they are charged a cost that depends on the number of players simultaneously using the same transition.

We study, from a complexity perspective, standard concepts of game theory in dynamic NCGs: social optima, Nash equilibria, and subgame perfect equilibria. Our contributions are the following: the existence of a strategy profile with social cost bounded by a constant is in PSPACE and NP-hard. (Pure) Nash equilibria always exist in dynamic NCGs; the existence of a Nash equilibrium with bounded cost can be decided in EXPSpace, and computing a witnessing strategy profile can be done in doubly-exponential time. The existence of a subgame perfect equilibrium with bounded cost can be decided in 2EXPSpace, and a witnessing strategy profile can be computed in triply-exponential time.

**2012 ACM Subject Classification** Theory of computation → Algorithmic game theory; Theory of computation → Formal languages and automata theory; Theory of computation → Solution concepts in game theory; Theory of computation → Verification by model checking

**Keywords and phrases** Congestion games, Nash equilibria, Subgame perfect equilibria, Complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.40

**Related Version** <https://arxiv.org/abs/2009.13632>.

## 1 Introduction

Congestion games model selfish resource sharing among several players [19]. A special case is the one of network congestion games, in which players aim at routing traffic through a congested network. Their popularity is certainly due to the fact that they have important practical applications, whether in transportation networks, or in large communication networks [18]. In network congestion games, each player chooses a set of transitions, forming a simple path from a source state to a target state, and the cost of a transition increases with its load, that is, with the number of players using it.

Network congestion games can be classified into atomic and non-atomic variants. Non-atomic semantics is appropriate for large populations of players, thus seen as a continuum. One then considers portions of the population that apply predefined strategies, and there



© Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 40; pp. 40:1–40:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is no such thing as the effect of an individual player on the cost of others. In contrast, in atomic games, the number of players is fixed, and each player may significantly impact the cost other players incur. We only focus on atomic games in this paper.

**Network congestion games.** Network congestion games, also called atomic selfish routing games in the literature, were first considered by Rosenthal [19]. These games are defined by a directed graph, a number of pairs of source-target vertices, and non-decreasing cost functions for each edge in the graph. For each source-target pair, a player must choose a route from the source to the target vertex. Given their choice of simple paths, the cost incurred by a player depends on the number of other players that choose paths sharing edges with their path, and on the cost functions of these edges. In this setting, a Nash equilibrium maps each player to a path in such a way that no player has an incentive to deviate: they cannot decrease their cost by choosing a different path.

Rosenthal proved that they are potential games, so that Nash equilibria always exist. Monderer and Shapley [16] studied in a more general way potential games, and explained how to iteratively use best-response strategies to converge to an equilibrium. Interestingly, under reasonable assumptions on the cost functions, Bertsekas and Tsitsiklis established that there is a direct correspondence between equilibria in selfish routing and distributed shortest-path routing, which is used in practice for packet routing in computer networks [7]. We refer the interested reader to [20] for an introduction and many basic results on general routing games.

A natural question is whether selfish routing is very different from a routing strategy decided by a centralized authority. In other words, how far can a selfish optimum be from the social optimum, in which players would cooperate. The notion of price of anarchy, first proposed by Koutsoupias and Papadimitriou [13], is the ratio of the worst cost of a Nash equilibrium and the cost of the social optimum. This measures how bad Nash equilibria can be. In the context of network congestion games, the price of anarchy was first studied by Suri *et al.* [21], establishing an upper bound of  $\frac{5}{2}$  when all cost functions are affine. A refined upper bound was provided by Awerbuch *et al.* [5]. Bounds on the dual notion of price of stability, which is the ratio of the cost of a best Nash equilibrium and the cost of the social optimum was also studied for routing games [1].

**Timing aspects.** Several works investigated refinements of this setting. In [10], the authors study network congestion games in which each edge is traversed with a fixed duration independent of its load, while the cost of each edge depends on the load. The model is thus said to have *time-dependent* costs since the load depends on the times at which players traverse a given edge. The authors prove the existence of Nash equilibria by reduction to the setting of [19]. An extension of this setting with timed constraints was studied in [2, 3].

The setting of fixed durations with time-dependent costs is interesting in applications where the players sharing a resource (an edge) see their quality of service decrease, while the time to use the resource is unaffected [3]. This might be the case, for instance, in some telecommunication and multimedia streaming applications. Timing also appears, for instance, in [17, 14] where the load affects travel times and players' objective is to minimize the total travel time. Other works focus on flow models with a timing aspect [12, 8].

**Dynamic network congestion games.** In classical network congestion games, including those mentioned above, players choose their strategies (*i.e.*, their *simple* paths) in one shot. However, it may be interesting to let agents choose their paths *dynamically*, that is, step by

step, by observing other players' previous choices. In this paper, we study network congestion games with time-dependent costs as in [10], but with unit delays, and in a dynamic setting. More precisely, at each step, each of the players simultaneously selects the edge they want to take; each player is then charged a cost that depends on the load of the edge they selected, and traverses that edge in one step. We name these games *dynamic network congestion games* (dynamic NCGs in short); the behaviour of the players in such games is formalized by means of *strategies*, telling the players what to play depending of the current configuration. Notice that, because congestion effect applies to edges used *simultaneously* by several players, taking cycles can be interesting in dynamic NCGs, which makes our setting more complex than most NCG models [4, 10, 19, 20].

Such a dynamic setting was studied in [4] for resource allocation games, which extends [19] with dynamic choices. A more detailed related work appears at the end of this section.

**Standard solution concepts.** We study classical solution concepts on dynamic network congestion games. A strategy profile (*i.e.*, a function assigning a strategy to each player) is a *Nash Equilibrium* (NE) when each single strategy is an optimal response to the strategies of the other players; in other terms, under such a strategy profile, no player may lower their costs by unilaterally changing their strategies. Notice that NEs need not exist in general, and when they exist, they may not be unique. In the setting of dynamic games, Nash Equilibria are usually enforced using *punishing strategies*, by which any deviating player will be punished by all other players once the deviation has been detected. However, such punishing strategies may also increase the cost incurred to the punishing players, and hence do not form a credible threat; *Subgame-Perfect Equilibria* (SPEs) refine NEs and address this issue by requiring that the strategy profile is an NE along any play.

NEs and SPEs aim at minimizing the individual cost of each player (without caring of the others' costs); in a collaborative setting, the players may instead try to lower the social cost, *i.e.*, the sum of the costs incurred to all the players. Strategy profiles achieving this are called *social optima* (SO). Obviously, the social cost of NEs and SPEs cannot be less than that of the social optimum; the *price of anarchy* measures how bad selfish behaviours may be compared to collaborative ones.

**Our contributions.** We take a computational-complexity viewpoint to study dynamic network congestion games. We first establish the complexity of computing the social optimum, which we show is in PSPACE and NP-hard. We then prove that best-response strategies can be computed in polynomial time, and that dynamic NCGs are potential games, thereby showing the existence of Nash equilibria in any dynamic NCG; this also shows that some Nash equilibrium can be computed in pseudo-polynomial time. We then give an EXPSPACE (resp. 2EXPSPACE) algorithm to decide the existence of Nash Equilibria (resp. Subgame-Perfect Equilibria) whose costs satisfy given bounds. This allows us to compute best and worst such equilibria, and then the price of anarchy and the price of stability.

Note that some of the high complexities follow from the binary encoding of the number of players, which is the main input parameter. For instance, the exponential-space complexity drops to pseudo-polynomial time for a fixed number of players. This parameter becomes important since we advocate the study of computational problems, such as computing Nash equilibria with a given cost bound. We also believe that computing precise values for price of anarchy and the price of stability is interesting, rather than providing bounds on the set of all instances as in *e.g.* [21].

Omitted proofs can be found in the corresponding arXiv article [6].

**Comparison with related work.** The works closest to our setting are [10, 4, 2, 3]. As in [10, 3], we establish the existence of Nash equilibria using potential games. Unlike [10], we cannot obtain this result immediately by reducing our games to congestion games [19] since the lengths of the strategies cannot be bounded *a priori*. Moreover, the best-response problem has a polynomial-time solution in our setting while the problem is NP-hard both in [10, 3]. In [10], this is due to the possibility of having arbitrary durations, while the source of complexity in [2, 3] is due to the use of timed automata. Thus, our setting offers a simpler way of expressing timings, and avoids their high complexity for this problem.

Dynamic choices were studied in [4] but with a different cost model. Moreover, network congestion games can only be reduced to such a setting given an a priori bound on the length of the paths. So we cannot directly transfer any of their results to our setting. Dynamic choices were also studied in [10] in the setting of coordination mechanisms which are local policies that allow one to sequentialize traffic on the edges.

## 2 Preliminaries

### 2.1 Dynamic network congestion games

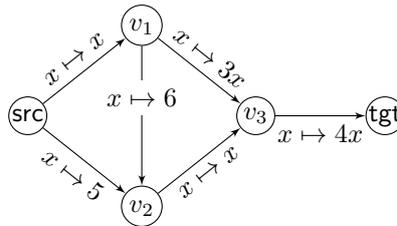
Let  $\mathcal{F}$  be the family of non-decreasing functions from  $\mathbb{N}$  to  $\mathbb{N}$  that are piecewise-affine, with finitely many pieces. We assume that each  $f \in \mathcal{F}$  is represented by the endpoints of intervals, and the coefficients, all encoded in binary. An arena for dynamic network congestion games is a weighted graph  $\mathcal{A} = \langle V, E, \text{src}, \text{tgt} \rangle$ , where  $V$  is a finite set of states,  $E: V \times V \rightarrow \mathcal{F}$  is a partial function defining the cost of edges, and  $\text{src}$  and  $\text{tgt}$  are a source- and a target state in  $V$ . It is assumed throughout this paper that  $\text{tgt}$  has only a single outgoing transition, which is a self-loop with constant cost function  $x \mapsto 0$ . We also assume that  $\text{tgt}$  is reachable from all other states.

A dynamic network congestion game (dynamic NCG for short) is a pair  $\mathcal{G} = \langle \mathcal{A}, n \rangle$  where  $\mathcal{A}$  is an arena as above and  $n \in \mathbb{N}$  is the (binary-encoded) number of players. In a dynamic network congestion game, all players start from  $\text{src}$  and simultaneously select the edges they want to traverse, with the aim of reaching the target state with minimal individual accumulated cost. Taking an edge  $e = (v, f, v')$  has a cost  $f(l)$ , where  $l$  is the number of players simultaneously using edge  $e$ . The cost function of edge  $e$  is denoted by  $f_e$ . We let  $\kappa = \max_{e \in E} f_e(n)$ , which is the maximal cost that a player may endure along one edge.

Our setting differs from classical network congestion games [20] mainly in two respects:

- first, the game is played in rounds, during which all players take exactly one transition; the number of players using an edge  $e$  is measured *dynamically*, at each round;
- second, during the play, players may adapt their choices to what the other players have been doing in the previous rounds.

► **Remark 1.** In this work, we mainly focus on the *symmetric* case, where all players have the same source and target. This is because we take a parametric-verification point of view, with



■ **Figure 1** Representation of an arena for a dynamic NCG (loop omitted on  $\text{tgt}$ ).

the (long-term) aim of checking properties of dynamic NCGs for arbitrarily many players. An important consequence of this choice is that the number of players now is encoded in binary, which results in an exponential blow-up in the number of configurations of the game (compared to the asymmetric setting).

**Semantics as a concurrent game.** For any  $k \in \mathbb{N}$ , we write  $\llbracket k \rrbracket = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$ . A *configuration* of a dynamic network congestion game  $\langle \mathcal{A}, n \rangle$  is a mapping  $c: \llbracket n \rrbracket \rightarrow V$ , indicating the position of each player in the arena. We define  $c_{\text{src}}: i \in \llbracket n \rrbracket \mapsto \text{src}$  and  $c_{\text{tgt}}: i \in \llbracket n \rrbracket \mapsto \text{tgt}$  as the initial and target configurations, respectively.

With  $\langle \mathcal{A}, n \rangle$ , we first associate a multi-weighted graph  $\mathcal{M} = \langle C, T \rangle$ , where  $C = V^{\llbracket n \rrbracket}$  is the set of all configurations and  $T \subseteq C \times \mathbb{N}^{\llbracket n \rrbracket} \times C$  is a set of edges, defined as follows: there is an edge  $(c, w, c')$  in  $T$  if, and only if, there exists a collection  $\mathbf{e} = (e_i)_{i \in \llbracket n \rrbracket}$  of edges of  $E$  such that for all  $i \in \llbracket n \rrbracket$ , writing  $e_i = (v_i, f_i, v'_i)$  and  $u_i = \#\{j \in \llbracket n \rrbracket \mid e_j = e_i\}$ , we have  $c(i) = v_i$ ,  $c'(i) = v'_i$ , and  $w(i) = f_i(u_i)$ . We denote this edge with  $c \xrightarrow{\mathbf{e}} c'$ . We may omit to mention  $\mathbf{e}$  since it can be obtained from  $c$  and  $c'$ ; similarly, we write  $\text{cost}_i(c, c')$  for  $w(i)$ .

Two edges  $(c, w, c')$  and  $(d, x, d')$ , in that order, are said to be *consecutive* whenever  $d = c'$ . Given a configuration  $c$ , a *path* from  $c$  in a dynamic network congestion game is either the single configuration  $c$  (we call this a trivial path) or a non-empty, finite or infinite sequence of consecutive edges  $\rho = (t_j)_{1 \leq j < |\rho|}$  in  $\mathcal{M}$ , where  $t_1$  is a transition from  $c$ ; the size of a path  $\rho$  is one for trivial paths, and  $|\rho| \in \mathbb{N} \cup \{+\infty\}$  otherwise. We write  $\text{Paths}(\langle \mathcal{A}, n \rangle, c)$  and  $\text{Paths}^\omega(\langle \mathcal{A}, n \rangle, c)$  for the set of finite and infinite paths from  $c$  in  $\langle \mathcal{A}, n \rangle$ , respectively.

With each path  $\rho = (c_j, w_j, c'_j)_j$ , and each player  $i \in \llbracket n \rrbracket$ , we associate a *cost*, written  $\text{cost}_i(\rho)$ , which is zero for trivial paths,  $+\infty$  for infinite paths along which  $c_j(i) \neq \text{tgt}$  for all  $j$ , and  $\sum_{j=1}^{|\rho|-1} w_j(i)$  otherwise. We define the *social cost* of  $\rho$ , denoted by  $\text{soccost}(\rho)$ , as  $\sum_{i \in \llbracket n \rrbracket} \text{cost}_i(\rho)$ .

Given a path  $\rho$ , an index  $1 \leq j < |\rho| + 1$  and a player  $i \in \llbracket n \rrbracket$ , we write  $\rho(j)$  for the  $j$ -th configuration of  $\rho$ , and  $\rho(j)(i)$  for the state of Player  $i$  in that configuration. For  $j \geq 2$ , we define  $\rho_{\leq j}$  as the prefix of  $\rho$  that ends in the  $j$ -th configuration; we let  $\rho_{\leq 1} = \rho(1)$ . Similarly, for  $1 \leq j \leq |\rho| - 1$ , we let  $\rho_{\geq j}$  denote the suffix that starts at the  $j$ -th configuration. Finally, if  $|\rho|$  is finite, we let  $\rho_{\geq |\rho|} = \rho(|\rho|)$ .

► **Example 2.** Consider the arena  $\mathcal{A}$  displayed at Fig. 1 and the dynamic NCG  $\langle \mathcal{A}, 2 \rangle$  with two players. Assume that Player 1 follows the path  $\pi_1: \text{src} \rightarrow v_1 \rightarrow v_3 \rightarrow \text{tgt}$  and Player 2 goes via  $\pi_2: \text{src} \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \text{tgt}$ . This gives rise to the following path:

$$\left( \begin{array}{l} 1 \mapsto \text{src} \\ 2 \mapsto \text{src} \end{array} \right) \xrightarrow{\begin{array}{l} 1 \mapsto 2 \\ 2 \mapsto 2 \end{array}} \left( \begin{array}{l} 1 \mapsto v_1 \\ 2 \mapsto v_1 \end{array} \right) \xrightarrow{\begin{array}{l} 1 \mapsto 3 \\ 2 \mapsto 6 \end{array}} \left( \begin{array}{l} 1 \mapsto v_3 \\ 2 \mapsto v_2 \end{array} \right) \xrightarrow{\begin{array}{l} 1 \mapsto 4 \\ 2 \mapsto 1 \end{array}} \left( \begin{array}{l} 1 \mapsto \text{tgt} \\ 2 \mapsto v_3 \end{array} \right) \xrightarrow{\begin{array}{l} 1 \mapsto 0 \\ 2 \mapsto 4 \end{array}} \left( \begin{array}{l} 1 \mapsto \text{tgt} \\ 2 \mapsto \text{tgt} \end{array} \right)$$

Notice how edge  $v_3 \rightarrow \text{tgt}$  of  $\mathcal{A}$  is used by both players, but not simultaneously, so that the cost of using that edge is 4 for each of them, while it would be 8 in classical NCGs. ◀

We now extend this graph to a concurrent game structure. A *move* for Player  $i \in \llbracket n \rrbracket$  from configuration  $c$  is an edge  $e = (v, f, v') \in E$  such that  $v = c(i)$ . A *move vector* from  $c$  is a sequence  $\mathbf{e} = (e_i)_{i \in \llbracket n \rrbracket}$  such that for all  $i \in \llbracket n \rrbracket$ ,  $e_i$  is a move for Player  $i$  from  $c$ .

A network congestion game  $\langle \mathcal{A}, n \rangle$  then gives rise to a concurrent game structure  $\mathcal{S} = \langle C, T, M, U \rangle$  where  $\langle C, T \rangle$  is the graph defined above,  $M: C \times \llbracket n \rrbracket \rightarrow 2^E$  lists the set of possible moves for each player in each configuration, and  $U: C \times E^{\llbracket n \rrbracket} \rightarrow T$  is the transition function, such that for every configuration  $c$  and every move vector  $\mathbf{e} = (e_i)_{i \in \llbracket n \rrbracket}$  with  $e_i \in M(c, i)$  for all  $i \in \llbracket n \rrbracket$ ,  $U(c, \mathbf{e}) = (c \xrightarrow{\mathbf{e}} c')$ .

A *strategy* for Player  $i$  in  $\mathcal{S}$  from configuration  $c$  is a function  $\sigma_i: \text{Paths}(\langle \mathcal{A}, n \rangle, c) \rightarrow E$  that associates, with any finite path  $\rho$  from  $c$  in  $\mathcal{S}$ , a move for this player from the last configuration of  $\rho$ . A *strategy profile* is a family  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$  of strategies, one for each player. We write  $\mathfrak{S}$  for the set of strategies, and  $\mathfrak{S}^n$  for the set of strategy profiles.

Let  $c$  be a configuration,  $h$  be a finite path from  $c$  and a strategy profile  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$  from  $c$ . The *residual strategy profile* of  $\sigma$  after  $h$  is the strategy profile  $\sigma^h = (\sigma_i^h)_{i \in \llbracket n \rrbracket}$  from the last configuration of  $h$  defined by  $\sigma_i^h(h') = \sigma_i(h \cdot h')$ , where  $h \cdot h'$  is the concatenation of paths  $h$  and  $h'$ .

The *outcome* of a strategy profile  $\sigma$  from  $c$  is the infinite path  $\rho = (c_i, w_i, c_{i+1})_{i \geq 1}$ , hereafter denoted with  $\text{outcome}(\sigma)$ , obtained by running the strategy profile; it is formally defined as the only infinite path such that  $(c_1, w_1, c_2) = U(c, \sigma(c))$ , and such that for any  $j \geq 2$ ,  $(c_j, w_j, c_{j+1}) = U(c_j, \sigma(h'))$ , where  $h' = (c_1, w_1, c_2) \cdots (c_{j-1}, w_{j-1}, c_j)$ .

Pick a strategy profile  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$ , and let  $\rho = (t_j)_{j \geq 1}$  be its outcome, writing  $t_j = (c_j, (w_j^i)_{i \in \llbracket n \rrbracket}, c'_j)$  for all  $j \geq 1$ . Let  $k \in \llbracket n \rrbracket$ . If  $c'_l(k) = \text{tgt}$  for some  $l \in \mathbb{N}$ , then  $\sigma_k$  is said to be winning for Player  $k$ . In that case, we define  $\text{cost}_k(\sigma)$  as  $\text{cost}_k(\text{outcome}(\sigma))$ . If  $c'_l(i) = \text{tgt}$  for all  $i \in \llbracket n \rrbracket$ , we define the *social cost* of  $\sigma$  as  $\text{soccost}(\sigma) = \text{soccost}(\rho)$ .

A strategy  $\sigma_i$  for Player  $i$  is said *blind* whenever for any two finite paths  $\rho$  and  $\rho'$  having same length  $k$ , if for any position  $0 \leq j < k$  we have  $\rho(j)(i) = \rho'(j)(i)$ , then  $\sigma_i(\rho) = \sigma_i(\rho')$ . Intuitively, this means that strategy  $\sigma_i$  follows a path in  $\mathcal{A}$ , independently of what the other players do. A blind strategy can thus be represented as a path and we write  $|\sigma_i|$  for the length of that path (until its first visit to  $\text{tgt}$ , if any). We write  $\mathfrak{B}$  for the set of blind strategies.

► **Example 3.** Consider again the arena  $\mathcal{A}$  of Fig. 1. The paths  $\pi_1$  and  $\pi_2$  from Example 2 are two blind strategies in that dynamic NCG. In a 2-player setting, an example of a non-blind strategy  $\sigma$  consists in first taking the transition  $\text{src} \rightarrow v_1$ , and then either taking  $v_1 \rightarrow v_3$  if the other player took the same initial transition, or taking  $v_1 \rightarrow v_2$  otherwise. ◀

**Representation as a weighted graph.** Another way of representing configurations is to consider their Parikh images. With a configuration  $c \in V^{\llbracket n \rrbracket}$ , we associate an abstract configuration  $\bar{c} \in \llbracket n \rrbracket^V$  defined as  $\bar{c}(v) = \#\{i \in \llbracket n \rrbracket \mid c(i) = v\}$ .

The *abstract weighted graph* associated with a dynamic network congestion game  $\langle \mathcal{A}, n \rangle$  is the weighted graph  $\mathcal{P} = \langle A, B \rangle$ , where  $A$  contains all abstract configurations, and there is an edge  $(a, w, a')$  in  $B \subseteq A \times \mathbb{N} \times A$  if, and only if, there is a mapping  $b: E \rightarrow \llbracket n \rrbracket$  such that  $\sum_{e \in E} b(e) = n$  and for all  $v \in V$ ,

$$a(v) = \sum_{e=(v,f,v')} b(e) \quad w = \sum_{e=(v,f,v')} b(e) \times f(b(e)) \quad a'(v) = \sum_{e=(v',f,v)} b(e).$$

Similarly to the representation as multi-weighted graphs, an *abstract path* of a network congestion game is either a single configuration or a non-empty, finite or infinite sequence of consecutive edges in the abstract weighted graph. The *cost* of an abstract path is the sum of the weights of its edges (if any). Then:

► **Lemma 4.** *For any  $w \in \mathbb{N} \cup \{+\infty\}$ , there is an abstract path in  $\mathcal{M}$  with social cost  $w$  if, and only if, there is an abstract path in  $\mathcal{P}$  with cost  $w$ .*

## 2.2 Social optima and equilibria

Consider a dynamic network congestion game  $\mathcal{G} = \langle \mathcal{A}, n \rangle$ . We recall standard ways of optimizing the strategies of the players, depending on the situation.

In a collaborative situation, all players want to collectively minimize the total cost for having all of them reach the target state of the arena. Formally, a strategy profile  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$  realizes the *social optimum* if  $\text{soccost}(\sigma) = \inf_{\tau \in \mathfrak{S}^n} \text{soccost}(\tau)$ .

In a selfish situation, each player aims at optimizing their response to the others' strategies. Given a strategy profile  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$ , a player  $k \in \llbracket n \rrbracket$ , and a strategy  $\sigma'_k \in \mathfrak{S}$ , we denote by  $\langle \sigma_{-k}, \sigma'_k \rangle$  the strategy profile  $(\tau_i)_{i \in \llbracket n \rrbracket}$  such that  $\tau_k = \sigma'_k$  and  $\tau_i = \sigma_i$  for all  $i \in \llbracket n \rrbracket \setminus \{k\}$ . The strategy  $\sigma_k$  is a *best response* to  $(\sigma_i)_{i \in \llbracket n \rrbracket \setminus \{k\}}$  if  $\text{cost}_k(\sigma) = \inf_{\sigma'_k \in \mathfrak{S}} \text{cost}_k(\langle \sigma_{-k}, \sigma'_k \rangle)$ . A strategy profile  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$  is a *Nash equilibrium* if for each  $k \in \llbracket n \rrbracket$ , the strategy  $\sigma_k$  is a best response to  $(\sigma_i)_{i \in \llbracket n \rrbracket \setminus \{k\}}$ . In such a case, no player has profitable unilateral deviations, *i.e.*, no player alone can decrease their cost by switching to a different strategy.

Nash equilibria can be defined for subclasses of strategy profiles. In particular, a *blind Nash equilibrium* is a blind strategy profile  $\sigma$  that is a Nash equilibrium *for blind-strategy deviations*: for all  $k \in \llbracket n \rrbracket$ ,  $\text{cost}_k(\sigma) = \inf_{\sigma'_k \in \mathfrak{B}} \text{cost}_k(\langle \sigma_{-k}, \sigma'_k \rangle)$ . *A priori*, a blind Nash equilibrium need not be a Nash equilibrium for general strategies.

In an NE, once a player deviated from their original strategy in the strategy profile, the other players can punish the deviating player, even if this results in increasing their own costs. Indeed, the condition for being an NE only requires that the deviation should not be profitable to the deviating player. Subgame-Perfect Equilibria (SPE) refine NEs and rule out such non-credible threats by requiring that, for any path  $h$  in the configuration graph, the residual strategy profile after  $h$  is an NE.

► **Example 5.** Consider again the dynamic NCG  $\langle \mathcal{A}, 2 \rangle$ , with the arena  $\mathcal{A}$  of Fig. 1. Assume that Player 1 plays the blind strategy corresponding to  $\pi_3: \text{src} \rightarrow v_2 \rightarrow v_3 \rightarrow \text{tgt}$ , while Player 2 plays the non-blind strategy  $\sigma$  of Example 3. The cost for Player 1 then is 10, while that of Player 2 is 12.

This strategy profile is an NE: Player 2 could be tempted to play  $\pi_1$ , but they would then synchronize with Player 1 on edge  $v_3 \rightarrow \text{tgt}$ , and get cost 12 again. Similarly, Player 1 could be tempted to play  $\pi_1$  instead of  $\pi_3$ , but in that case strategy  $\sigma$  would tell Player 2 to follow the same path, and the cost for Player 1 (and 2) would be 16. Notice in particular that this is not an SPE, but that the blind strategy profile  $\langle \pi_1, \pi_2 \rangle$  (extended to the whole configuration tree in the only possible way) is an SPE in  $\langle \mathcal{A}, 2 \rangle$ . ◀

In Sections 4 and 5, we focus on NEs and SPEs, developing EXPSPACE and 2EXPSPACE-algorithms for deciding the existence of NEs and SPEs respectively of social cost less than or equal to a given bound. Actually, our approach extends to the  $\vec{\gamma}$ -weighted social cost, where  $\vec{\gamma} \in \mathbb{Z}^{\llbracket n \rrbracket}$  are coefficients applied to the costs of the respective players when computing the social cost. As a consequence, we can compute best and worst NEs and SPEs, hence also the price of anarchy and price of stability [13]. Before that, in Section 3, we extend classical techniques using blind strategies to compute the social optimum and prove that NEs always exist.

### 3 Socially-optimal strategy profiles

To compute a socially-optimal strategy profile, it suffices to find a path in the concurrent game structure of the given network congestion game with minimal total cost since one can define a strategy profile that induces any given path. Rather than finding such a path in the concurrent game structure, and in view of Lemma 4, one can look for one in the abstract weighted graph, thereby reducing in complexity. The socially-optimal cost in a dynamic NCG  $\langle \mathcal{A}, n \rangle$  is thus the cost of a shortest path in the associated weighted abstract graph  $\mathcal{P}$  from  $\bar{c}_{\text{src}}$  to  $\bar{c}_{\text{tgt}}$ .

Since  $\mathcal{P}$  has exponential size, we derive complexity upper bounds for computing a socially-optimal strategy and deciding the associated decision problem. Moreover, adapting [15, Theorem 4.1] which proves NP-hardness in classical NCGs, we provide a reduction from the Partition problem to establish an NP lower-bound.

► **Theorem 6.** *A socially-optimal strategy profile can be computed in exponential time. The constrained social-optimum problem is in PSPACE and NP-hard.*

Note, that while  $\mathcal{P}$  has size  $(n+1)^{|V|}$ , it is sufficient to consider paths with a smaller number of transitions when looking for a shortest path:

► **Lemma 7.** *There is a shortest path (w.r.t. cost) in  $\mathcal{P}$  with size (in terms of its number of transitions) at most  $n \cdot |V|$ .*

► **Remark 8.** A consequence of Lemma 7 is that deciding the constrained social-optimum problem is in NP for asymmetric games, since in that setting the lists of sources and targets of each player is part of the input, so that  $n$  is polynomial in the size of the input. However, our NP-hardness proof only works in the symmetric case.

## 4 Nash equilibria

In this section, we study the existence of Nash equilibria and give algorithms to compute them under given constraints.

### 4.1 Existence and computation of (blind) Nash equilibria

To prove that blind Nash equilibria always exist, we establish that dynamic NCGs with blind strategies are potential games [19, 16] which are known to have Nash equilibria.

Consider a dynamic NCG  $\langle \mathcal{A}, n \rangle$ , a blind strategy profile  $\pi$ , and let  $N_\pi$  denote the maximum length of the paths prescribed by  $\pi$ . We define the following potential function, which is an adaptation of that used in [19]:

$$\psi(\pi) = \sum_{j=1}^{N_\pi} \sum_{e \in E} \sum_{i=1}^{\text{load}_e(\pi, j)} f_e(i),$$

where  $\text{load}_e(\pi, j)$  denotes the number of players that take edge  $e$  in the  $j$ -step under  $\pi$ , and  $f_e$  is the cost function on edge  $e$ .

Using the above-defined potential function, one can derive an algorithm to find a Nash equilibrium, by a classical *best-response* iteration. Starting with an arbitrary blind strategy profile, at each step we replace some player's strategy with their best-response, and we continue as long as some player's cost can be decreased. When this procedure terminates, the profile at hand is a blind Nash equilibrium. In dynamic NCGs, best responses exist and can be computed in polynomial time. Indeed, one can construct a game in which all players but Player  $i$  follow their fixed strategies given by profile  $\pi$ , using  $N_\pi$  copies of the game in order to distinguish the steps. After the  $N_\pi$ -step, all players in  $\llbracket n \rrbracket \setminus \{i\}$  have reached their targets. Since it is the only remaining player, the remaining path for Player  $i$  should not be longer than  $|V|$ . Altogether, we obtain the following complexity upper-bound:

► **Theorem 9.** *In dynamic NCGs, blind Nash equilibria always exist, and we can compute one in pseudo-polynomial time.*

► **Remark 10.** As an alternative proof to existence of blind NEs, we could have bounded the length of outcomes of blind NEs as follows: all players have a strategy realizing cost at most  $|V| \cdot \kappa$ , where  $\kappa = \max_{e \in E} f_e(n)$ , since the shortest path from **src** to **tgt** has length at most  $|V|$ , and the cost for a player at each step along edge  $e$  is at most  $\kappa$ . It follows that no path along which the cost for some player is larger than  $|V| \cdot \kappa$  can be the outcome of a blind NE. As a consequence, if a dynamic NCG has a blind NE, then it has one of length at most  $|V| \cdot \kappa \cdot |V|^n$  (by removing zero-cycles). Using this bound, we can transform dynamic NCGs into classical congestion games, in which blind NEs always exist [10, 19].

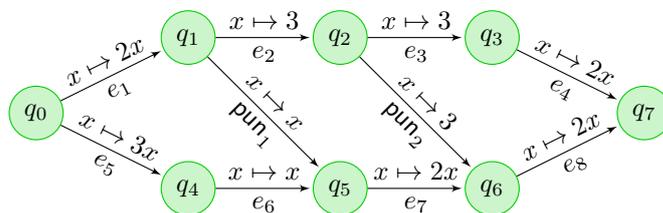
We now show that blind Nash equilibria are in fact Nash equilibria. This is proved using the observation that given a blind strategy profile, the most profitable deviation for any player can be assumed to be a blind strategy.

► **Lemma 11.** *In dynamic NCGs, blind Nash equilibria are Nash equilibria.*

Computing some (blind) Nash equilibrium may not be satisfactory for two reasons: one might want to compute the best (or the worst) Nash equilibrium in terms of the social cost; and as Lemma 12 claims, blind Nash equilibria are suboptimal, *i.e.*, a lower social cost can be achieved by Nash equilibria with general strategies. This justifies the study of more complex strategy profiles in the next subsection.

► **Lemma 12.** *There exists a dynamic NCG with a Nash equilibrium  $\pi$  such that for all blind Nash equilibria  $\pi'$ , we have  $\text{cost}(\pi) < \text{cost}(\pi')$ .*

The proof is based on the dynamic NCG depicted on Fig. 2, for which we prove there is a Nash equilibrium with total cost 36, while any *blind* Nash equilibrium has higher social cost.



■ **Figure 2** An arena on which blind Nash equilibria are sub-optimal.

## 4.2 Computation of general Nash equilibria

**Characterization of outcomes of Nash Equilibria.** Let us consider a dynamic NCG  $\langle \mathcal{A}, n \rangle$ , and the corresponding game structure  $\mathcal{S} = \langle C, T, M, U \rangle$ . Given two configurations  $c, c'$  with  $c \Rightarrow c'$ , we let  $\text{cost}_i(c, c')$  denote the cost of Player  $i$  on this transition from  $c(i)$  to  $c'(i)$ . We define  $\text{dev}_i(c, c')$  as the set of all configurations reachable when all players but Player  $i$  choose moves prescribed by the given transition  $c \Rightarrow c'$ :

$$\text{dev}_i(c, c') = \{c'' \in C \mid c \Rightarrow c'' \text{ and } \forall j \in \llbracket n \rrbracket \setminus \{i\}. c''(j) = c'(j)\}.$$

The *value* of configuration  $c$  for Player  $i$  is  $\text{val}_{i,c} = \sup_{\sigma_{-i} \in \mathcal{S}^{n-1}} \inf_{\sigma_i \in \mathcal{S}} \text{cost}_i((\sigma_{-i}, \sigma_i), c)$ . Note that the value corresponds to the value of the zero-sum game where Player  $i$  plays against the opposing coalition, starting at  $c$ . By [11], those values can be computed in polynomial time in the size of the game. Here the game is a 2-player game with state space  $|V| \times \llbracket n-1 \rrbracket^{|V|}$ , keeping track of the position of Player  $i$  and the abstract position of the

## 40:10 Dynamic Network Congestion Games

coalition. It follows that each  $\text{val}_{i,c}$  can be computed in exponential time in the size of the input  $\langle \mathcal{A}, n \rangle$ . Moreover, memoryless optimal strategies exist (in  $\mathcal{S}$ ), that is, the opposing coalition has a memoryless strategy  $\sigma_{-i}$  to ensure a cost of at least  $\text{val}_{i,c}$  from  $c$ .

The characterization of Nash equilibria outcomes is given in the following lemma.

► **Lemma 13.** *A path  $\rho$  in  $\langle \mathcal{A}, n \rangle$  is the outcome of a Nash equilibrium if, and only if,*

$$\forall i \in \llbracket n \rrbracket. \forall 1 \leq l < |\rho|. \forall c \in \text{dev}_i(\rho(l), \rho(l+1)). \quad \text{cost}_i(\rho_{\geq l}) \leq \text{val}_{i,c} + \text{cost}_i(\rho(l), c).$$

The intuition is that if the suffix  $\text{cost}_i(\rho_{\geq l})$  of  $\rho$  has cost more than  $\text{val}_{i,c} + \text{cost}_i(\rho(l), c)$ , then Player  $i$  has a profitable deviation regardless of the strategy of the opposing coalition, since  $\text{val}_{i,c}$  is the maximum cost that the coalition can inflict to Player  $i$  at configuration  $c$  where the deviation is observed. The lemma shows that the absence of such a suffix means that a Nash equilibrium with given outcome exists, which the proof constructs.

**Proof.** Consider a Nash equilibrium  $\sigma = (\sigma_i)_{i \in \llbracket n \rrbracket}$  with outcome  $\rho$ . Consider any player  $i$ , and any strategy  $\sigma'_i$  for this player. Let  $\rho'$  denote the outcome of  $\sigma[i \rightarrow \sigma'_i]$ . Let  $l$  denote the index of the last configuration where  $\rho$  and  $\rho'$  are identical. Since  $\sigma$  is a Nash equilibrium, we have  $\text{cost}_i(\rho) \leq \text{cost}_i(\rho')$ , that is,

$$\text{cost}_i(\rho_{\geq l}) \leq \text{cost}_i(\rho(l), \rho'(l+1)) + \text{cost}_i(\sigma[i \rightarrow \sigma'_i], \rho'_{\leq l+1})$$

where  $\text{cost}_i(\sigma[i \rightarrow \sigma'_i], \rho'_{\leq l+1})$  is the cost for Player  $i$  of the outcome of the residual strategy  $(\sigma[i \rightarrow \sigma'_i])^{\rho'_{\leq l+1}}$ . Since the choice of  $\sigma'_i$  is arbitrary here, we have,

$$\text{cost}_i(\rho_{\geq l}) \leq \text{cost}_i(\rho(l), \rho'(l+1)) + \inf_{\sigma'_i \in \mathfrak{S}} \text{cost}_i(\sigma[i \rightarrow \sigma'_i], \rho'_{\leq l+1}).$$

Moreover, we have  $\inf_{\sigma'_i \in \mathfrak{S}} \text{cost}_i(\pi[i \rightarrow \sigma'_i], \rho'_{\leq l+1}) = \inf_{\sigma'_i \in \mathfrak{S}} \text{cost}_i(\pi[i \rightarrow \sigma'_i], \rho'(l+1))$  since memoryless strategies suffice to minimize the cost [11]. We then have

$$\inf_{\sigma'_i \in \mathfrak{S}} \text{cost}_i(\pi[i \rightarrow \sigma'_i], \rho'(l+1)) \leq \sup_{\sigma_{-i} \in \mathfrak{S}^{n-1}} \inf_{\sigma_i \in \mathfrak{S}} \text{cost}_i((\sigma_{-i}, \sigma_i), \rho'(l+1)).$$

We obtain the required inequality

$$\begin{aligned} \text{cost}_i(\rho_{\geq l}) &\leq \text{cost}_i(\rho(l), \rho'(l+1)) + \sup_{\sigma_{-i} \in \mathfrak{S}^{n-1}} \inf_{\sigma_i \in \mathfrak{S}} \text{cost}_i((\sigma_{-i}, \sigma_i), \rho'(l+1)) \\ &\leq \text{cost}_i(\rho(l), c) + \text{val}_{i,c}. \end{aligned}$$

Conversely, consider a path  $\rho$  that satisfies the condition. We are going to construct a Nash equilibrium having outcome  $\rho$ . The idea is that players will follow  $\rho$ , and if some player  $i$  deviates, then the coalition  $-i$  will apply a joint strategy to maximize the cost of Player  $i$ , thus achieving at least  $\text{val}_{i,c}$ , where  $c$  is the first configuration where deviation is detected.

Let us define the punishment function  $\mathcal{P}_\rho: \text{Paths}(\langle \mathcal{A}, n \rangle) \rightarrow \llbracket n \rrbracket \cup \{\perp\}$  which keeps track of the deviating players and the step where such a player has deviated. For path  $h' = h(c, w, c')$ , we write

$$\mathcal{P}_\rho(h') = \begin{cases} \perp & \text{if } h' \leq_{\text{pref}} \rho, \\ i & \text{if } h \leq_{\text{pref}} \rho, h(c, w, c') \not\leq_{\text{pref}} \rho, \text{ and } i \in \llbracket n \rrbracket \text{ min. s.t. } c'(i) \neq \rho(|h|+1)(i), \\ \mathcal{P}_\rho(h) & \text{otherwise.} \end{cases}$$

Intuitively,  $\perp$  means that no players have deviated from  $\rho$  in the current path. If  $\mathcal{P}_\pi(h) = j$ , then Player  $j$  was among the first players to deviate from  $\rho$  in the path  $h$ ; so for some  $l$ ,

$h(l)(j) = \rho(l)(j)$  but  $h(l+1)(j) \neq \rho(l+1)(j)$ . Notice that if several players deviate at the same step, there are no conditions to be checked, and the strategy can be chosen arbitrarily. For each configuration  $c$  and coalition  $-i$ , let  $\sigma_{-i,c}$  be the strategy of coalition  $-i$  maximizing the cost of Player  $i$  from configuration  $c$ ; thus achieving at least  $\text{val}_{i,c}$ . Player  $j$ 's strategy in this coalition, for  $j \neq i$ , is denoted  $\sigma_{-i,c,j}$ . For path  $h' = h(c, w, c')$ , define

$$\tau_i(h') = \begin{cases} (c'(i), m(i), c''(i)) & \text{if } \mathcal{P}_\rho(h') = \perp, \rho(|h'| + 1) = (c', w', c''), \\ & \text{and } m \in E^n \text{ is such that } T(c', m) = (w', c''), \\ \text{arbitrary} & \text{if } \mathcal{P}_\rho(h') = i, \\ \sigma_{-j,c,i}(h') & \text{if } \mathcal{P}_\rho(h') = j \text{ for some } j \neq i. \end{cases}$$

The first case ensures that the outcome of the profile  $(\tau_i)_{i \in \llbracket n \rrbracket}$  is  $\rho$ . The third case means that Player  $i$  follows the coalition strategy  $\sigma_{-j,c}$  after Player  $j$  has deviated to configuration  $c$ . The second case corresponds to the case where Player  $i$  has deviated: the precise definition of this part of the strategy is irrelevant.

Let us show that this profile is indeed a Nash equilibrium. Consider any player  $j \in \llbracket n \rrbracket$  and any strategy  $\tau'_j$ . Let  $\rho'$  denote the outcome of  $(\tau_{-j}, \tau'_j)$ , and  $l$  the index of the last configuration where  $\rho$  and  $\rho'$  are identical. We have

$$\begin{aligned} \text{cost}_j((\tau_{-j}, \tau'_j)) &= \text{cost}_j(\rho_{\leq l}) + \text{cost}_j(\rho(l), \rho'(l+1)) + \text{cost}_j((\tau_{-j}, \tau_j), \rho'_{\leq l+1}) \\ &\geq \text{cost}_j(\rho_{\leq l}) + \text{cost}_j(\rho(l), \rho'(l+1)) + \text{val}_{j,\rho'(l+1)(j)} \\ &\geq \text{cost}_j((\tau_i)_{i \in \llbracket n \rrbracket}), \end{aligned}$$

where the second line follows from the fact that the coalition switches to a strategies ensuring a cost of at least  $\text{val}_{j,\rho'(l+1)(j)}$  at step  $l$ ; and the third line is obtained by assumption. This shows that  $(\tau_i)_{i \in \llbracket n \rrbracket}$  is indeed a Nash equilibrium and concludes the proof.  $\blacktriangleleft$

**Algorithm.** We define a graph that describes the set of outcomes of Nash equilibria by augmenting the  $n$ -weighted configuration graph  $\mathcal{M} = \langle C, T \rangle$ . For any real vector  $\vec{\gamma} = (\gamma_i)_{i \in \llbracket n \rrbracket}$ , we define the weighted graph  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}} = \langle C', T' \rangle$  with  $C' = C \times (\llbracket Y \rrbracket \cup \{0, \infty\})^n$  where  $Y = |V| \cdot \kappa$ , and  $T' \subseteq C' \times \mathbb{N} \times C'$ ; remember that all players have a strategy realizing cost at most  $Y$  in  $\langle \mathcal{A}, n \rangle$ . The initial state is  $(c_{\text{src}}, \infty^n)$ . The set of transitions  $T'$  is defined as follows:  $((c, b), z, (c', b')) \in T'$  if, and only if, there exists  $(c, w, c') \in T$ ,  $z = \vec{\gamma} \cdot w$  (where  $\cdot$  is dot product), and for all  $i \in \llbracket n \rrbracket$ ,

$$b'_i = \min(b_i - w_i, \min_{c'' \in \text{dev}_i(c(i), c'(i))} \text{cost}_i(c, c'') + \text{val}_{i,c''} - w_i). \quad (1)$$

Notice that by definition of  $C'$ ,  $b'_i$  must be nonnegative for all  $i \in \llbracket n \rrbracket$ , so there are no transitions  $((c, b), z, (c', b'))$  if the above expression is negative for some  $i$ . Notice also that the size of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$  is doubly-exponential in that of the input  $\langle \mathcal{A}, n \rangle$ , since this is already the case for  $C$ , while  $Y$  is singly-exponential.

Intuitively, for any path  $\rho$  that visits some state  $(c, b)$  in this graph, in order for  $\rho$  to be compatible with a Nash equilibrium, each player  $i$  must have cost no more than  $b_i$  in the rest of the path. In fact, the second term of the minimum in (1) is the least cost Player  $i$  could guarantee by not following  $(c, w, c')$  but going to some other configuration  $c'' \in \text{dev}_i(c, c')$ , so the bound  $b_i$  is used to guarantee that these deviations are not profitable. The definition of  $b'_i$  in (1) is the minimum of  $b_i - w_i$  and the aforementioned quantity since we check both the previous bound  $b_i$ , updated with the current cost  $w_i$  (which gives the left term), and the non-profitability of a deviation at the previous state (which is the right term). If this

## 40:12 Dynamic Network Congestion Games

minimum becomes negative, this precisely means that at an earlier point in the current path, there was a strategy for Player  $i$  which was more profitable than the current path regardless of the strategies of other players; so the current path cannot be the outcome of a Nash equilibrium. This is why the definition of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$  restricts the state space to nonnegative values for the  $b_i$ .

We prove that computing the cost of a Nash equilibrium minimizing the  $\vec{\gamma}$ -weighted social cost reduces to computing a shortest path in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$ . In particular, letting  $\gamma_i = 1$  for all  $i \in \llbracket n \rrbracket$ , a  $\vec{\gamma}$ -minimal Nash equilibrium is a best Nash equilibrium (minimizing the social cost), while taking  $\gamma_i = -1$  for all  $i \in \llbracket n \rrbracket$ , we get a worst Nash equilibrium (maximizing the social cost).

► **Theorem 14.** *For any dynamic NCG  $\langle \mathcal{A}, n \rangle$  and vector  $\vec{\gamma}$ , the cost of the shortest path from  $(c_{\text{src}}, \infty^n)$  to some  $(c_{\text{tgt}}, b)$  in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$  is the cost of a  $\vec{\gamma}$ -minimal Nash equilibrium.*

**Proof.** We show that for each path of  $\langle \mathcal{A}, n \rangle$  from  $c_{\text{src}}$  to  $c_{\text{tgt}}$ , there is a path in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$  from  $(c_{\text{src}}, \infty^n)$  to some  $(c_{\text{tgt}}, b)$  with the same cost, and vice versa.

Consider a Nash equilibrium  $\pi = (\sigma_j)_{j \in \llbracket n \rrbracket}$  with outcome  $\rho = (c_j, w_j, c_{j+1})_{1 \leq j < l}$ . We build a sequence  $b_1, b_2, \dots$  such that  $\rho' = ((c_j, b_j), \vec{\gamma} \cdot w_j, (c_{j+1}, b_{j+1}))_{1 \leq j < l}$  is a path of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$ . We set  $b_1(j) = \infty$  for all  $j \in \llbracket n \rrbracket$ . For  $j \geq 1$ , define

$$b_{j+1}(i) = \min \left( b_j(i) - w_j(i), \min_{c'' \in \text{dev}_j(c_j(i), c_{j+1}(i))} \text{cost}_i(c_j, c'') + \text{val}_{i, c''} - w_j(i) \right).$$

We are going to show that for all  $1 \leq j \leq l$ ,  $\text{cost}_i(\rho_{\geq j}) \leq b_j$ , which shows that  $b_j \geq 0$ , and thus  $\rho'$  is a path of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$ .

We show this by induction on  $j$ . This is clear for  $j = 1$ . Assume this holds up to  $j \geq 1$ . We have, by induction that  $\text{cost}_i(\rho_{\geq j}) \leq b_j(i)$  for all  $i \in \llbracket n \rrbracket$ . Moreover, since  $\pi$  is a Nash equilibrium, by Lemma 13,

$$\forall i \in \llbracket n \rrbracket, \text{cost}_i(\rho_{\geq j}) \leq \min_{c'' \in \text{dev}_i(\rho(j), \rho(j+1))} \text{val}_{i, c''} + \text{cost}_i(\rho(j), c'').$$

Therefore,

$$\begin{aligned} \text{cost}_i(\rho_{\geq j+1}) &= \text{cost}_i(\rho_{\geq j}) - w_j(i) \\ &\leq \min(b_j(i) - w_j(i), \min_{c'' \in \text{dev}_i(\rho(j), \rho(j+1))} \text{val}_{i, c''} + \text{cost}_i(\rho(j), c'') - w_j(i)) \end{aligned}$$

as required, and both paths have the same  $\vec{\gamma}$ -weighted cost.

Consider now a path  $((c_i, b_i), z_i, (c_{i+1}, b_{i+1}))_{1 \leq i < l}$  in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$ . By the definition of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \vec{\gamma}}$ , there exists  $w_1, w_2, \dots$  such that  $\rho = (c_j, w_j, c_{j+1})_{1 \leq j < l}$  is a path of  $\langle \mathcal{A}, n \rangle$ , and  $z_j = \vec{\gamma} \cdot w_j$ . So it only remains to show that  $\rho$  is the outcome of a Nash equilibrium. We will show that  $\rho$  satisfies the criterion of Lemma 13. We show by backwards induction on  $1 \leq j \leq l$  that for all  $i \in \llbracket n \rrbracket$ ,

1.  $\text{cost}_i(\rho_{\geq j}) \leq b_j(i)$ ,
2.  $\text{cost}_i(\rho_{\geq j}) \leq \min_{c'' \in \text{dev}_i(\rho(j), c'')} \text{cost}_i(\rho(j), c'') + \text{val}_{i, c''}$ .

For  $j = l$ , we have  $\text{cost}_i(\rho_{\geq l}) = 0$  so this is trivial. Assume the property holds down to  $j + 1$  for some  $1 \leq j < l$ . By induction hypothesis, we have

$$\text{cost}_i(\rho_{\geq j+1}) \leq b_{j+1}(i) = \min \left( b_j(i) - w_j(i), \min_{c'' \in \text{dev}_i(\rho(j), c'')} \text{cost}_i(\rho(j), c'') + \text{val}_{i, c''} - w_j(i) \right).$$

Therefore,

$$\text{cost}_i(\rho_{\geq j}) = \text{cost}_i(\rho_{\geq j+1}) + w_j(i) \leq \min \left( b_j(i), \min_{c'' \in \text{dev}_i(\rho(j), c'')} \text{cost}_i(\rho(j), c'') + \text{val}_{i, c''} \right),$$

as required. By Lemma 13,  $\rho$  is the outcome of a Nash equilibrium. ◀

Thanks to Theorem 14, we can compute the costs of the best and worst NEs of  $\langle \mathcal{A}, n \rangle$  in exponential space. We can also decide the existence of an NE with constraints on the costs (both social and individual), by non-deterministically guessing an outcome and checking in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \bar{\gamma}}$  that it is indeed an NE. We obtain the following conclusion:

► **Corollary 15.** *In dynamic NCGs, the constrained Nash-equilibrium problem is in EX-PSPACE.*

**Proof.** As noted earlier, the number of vertices in  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \bar{\gamma}}$  is doubly exponential since  $|C| = |V|^n$  is doubly exponential. Storing a configuration and computing its successors can be performed in exponential space. One can thus guess a path of size at most the size of the graph and check whether its cost is less than the given bound. This can be done using exponential-space counters, and provides us with an EXPSPACE algorithm. ◀

Note that one can effectively compute a Nash-equilibrium strategy profile satisfying the constraints in doubly-exponential time by finding the shortest path of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \bar{\gamma}}$ , and applying the construction of (the proof of) Lemma 13.

► **Remark 16.** The exponential complexity is due to the encoding of the number of players in binary. If we consider asymmetric NCGs, in which the source-target pairs would be given explicitly for all players, the size of  $\mathcal{G}_{\langle \mathcal{A}, n \rangle, \bar{\gamma}}$  would be singly-exponential, and the constrained Nash-equilibrium problem would be in PSPACE.

## 5 Subgame-perfect equilibria

In this section, we characterize the outcomes of SPEs and decide the existence of SPEs with constraints on the social cost. We follow the approach of [9], extending it to the setting of concurrent weighted games, which we need to handle dynamic NCGs.

**Characterization of outcomes of SPE.** Consider a dynamic NCG  $\langle \mathcal{A}, n \rangle$ , and the associated configuration graph  $\mathcal{M} = \langle C, T \rangle$ . We partition the set  $C$  of configurations into  $(X_j)_{0 \leq j \leq n}$  such that a configuration  $c$  is in  $X_j$  if, and only if,  $j = \#\{i \in \llbracket n \rrbracket \mid c(i) = \mathbf{tgt}\}$ . Since  $\mathbf{tgt}$  is a sink state in  $\mathcal{A}$ , if there is a transition from some configuration in  $X_j$  to some configuration in  $X_k$ , then  $k \geq j$ . We define  $X_{\geq j} = \bigcup_{i \geq j} X_i$ ,  $Z_j = \{(c, w, c') \in T \mid c \in X_j\}$  and  $Z_{\geq j} = \{(c, w, c') \in T \mid c \in X_{\geq j}\}$ .

Following [9], we inductively define a sequence  $(\lambda^{j*})_{0 \leq j \leq n}$ , where each  $\lambda^{j*} = \langle \lambda_i^{j*} \rangle_{i \in \llbracket n \rrbracket}$  is a  $n$ -tuple of labeling functions  $\lambda_i^{j*} : Z_{\geq j} \rightarrow \mathbb{N} \cup \{-\infty, +\infty\}$ . This sequence will be used to characterize outcomes of SPEs through the notion of  $\lambda$ -consistency:

► **Definition 17.** *Let  $j \leq n$ , and  $\lambda = (\lambda_i)_{i \in \llbracket n \rrbracket}$  be a family of functions such that  $\lambda : Z_{\geq j} \rightarrow \mathbb{N} \cup \{-\infty, +\infty\}$ . Let  $c \in X_{\geq j}$ . A finite path  $\rho = (t_k)_{1 \leq k < |\rho|}$  from  $c$  ending in  $c_{\mathbf{tgt}}$  is said to be  $\lambda$ -consistent whenever for any  $i \in \llbracket n \rrbracket$  and any  $1 \leq k < |\rho|$ , it holds  $\mathbf{cost}_i(\rho_{\geq k}) \leq \lambda_i(t_k)$ . We write  $\Gamma_\lambda(c)$  for the set of all  $\lambda$ -consistent paths from  $c$ .*

We now define  $\lambda^{j*}$  for all  $0 \leq j \leq n$  in such a way that, for all  $c \in X_{\geq j}$ ,  $\Gamma_{\lambda^{j*}}(c)$  is the set of all outcomes of SPEs in the subgame rooted at  $c$ . The case where  $j = n$  is simple: we have  $X_{\geq n} = \{c_{\mathbf{tgt}}\}$  and  $Z_{\geq n} = \{(c_{\mathbf{tgt}}, 0^n, c_{\mathbf{tgt}})\}$ ; there is a single path, which obviously is the outcome of an SPE since no deviations are possible. For all  $i \in \llbracket n \rrbracket$ , we let  $\lambda_i^{n*}(c_{\mathbf{tgt}}, 0^n, c_{\mathbf{tgt}}) = 0$ .

Now, fix  $j < n$ , assuming that  $\lambda^{(j+1)*}$  has been defined. In order to define  $\lambda^{j*}$ , we introduce an intermediary sequence  $(\mu_i^k)_{k \geq 0, i \in \llbracket n \rrbracket}$ , with  $\mu_i^k : Z_{\geq j} \rightarrow \mathbb{N} \cup \{-\infty, +\infty\}$ , of which  $(\lambda_i^{j*})_{i \in \llbracket n \rrbracket}$  will be the limit.

Functions  $\mu_i^k$  mainly operate on  $Z_j = Z_{\geq j} \setminus Z_{\geq j+1}$ : for any  $\mathbf{e} \in Z_{\geq j+1}$ , we let  $\mu_i^k(\mathbf{e}) = \lambda_i^{(j+1)*}(\mathbf{e})$ . Now, for  $\mathbf{e} = (c, w, c') \in Z_j$ ,  $\mu_i^k(\mathbf{e})$  is defined inductively as follows:

- $\mu_i^0(\mathbf{e}) = 0$  if  $c(i) = \text{tgt}$ , and  $\mu_i^0(\mathbf{e}) = +\infty$  otherwise;
- for  $k > 0$ ,  $\mu_i^k$  is defined from  $\mu_i^{k-1}$  following three cases: if  $c(i) = \text{tgt}$ , then  $\mu_i^k(\mathbf{e}) = 0$ ; if  $\Gamma_{\mu^{k-1}}(c') = \emptyset$  for some  $(c, w', c') \in T$ , then  $\mu_i^k(\mathbf{e}) = -\infty$ ; otherwise,

$$\mu_i^k(\mathbf{e}) = \min_{c'' \in \text{dev}_i(c, c')} \sup_{\rho \in \Gamma_{\mu^{k-1}}(c'')} (\text{cost}_i(c, c'') + \text{cost}_i(\rho))$$

We can then prove that for any  $e \in Z_{\geq j}$  and any  $k > 0$ ,  $\mu_i^k(e) \geq \mu_i^{k-1}(e)$ . It follows that the sequence  $(\mu^k)_{k \geq 0}$  stabilizes, and we can define  $\lambda^{j*}$  as its limit. Let  $\Gamma^* = \Gamma_{\lambda^{0*}}$ . Then:

► **Theorem 18.** *A path  $\rho$  in  $\mathcal{G} = \langle \mathcal{A}, n \rangle$  is the outcome of an SPE if, and only if,  $\rho \in \Gamma^*(c_{\text{src}})$ .*

**Algorithm.** It remains to compute the sequence  $(\mu^k)_{k \geq 0}$  (which will include checking non-emptiness of the corresponding  $\Gamma$ -sets), and to bound the stabilization time. To this aim, with any family  $\mu = (\mu_i)_{i \in \llbracket n \rrbracket}$  of functions as above and any configuration  $c$ , we associate an infinite-state counter graph  $\mathbb{C}[\mu, c] = \langle C', T' \rangle$  to capture all  $\mu$ -consistent paths from  $c$ :

- the set of vertices is  $C' = C \times (\mathbb{N} \cup \{+\infty\})^{\llbracket n \rrbracket}$ ;
- $T'$  contains all edges  $((d, b), w, (d', b'))$  for which  $(d, w, d')$  is an edge of  $\mathcal{M}$  and for all  $i \in \llbracket n \rrbracket$ ,  $b'(i) = 0$  if  $d(i) = \text{tgt}$ , and  $b'_i = \min\{b_i - w_i, \mu_i(d, w, d') - w_i\}$  otherwise (provided that  $b'_i \geq 0$  for all  $i$ , in order for  $(d', b')$  to be an edge of  $\mathbb{C}[\mu, c]$ ).

With the initial configuration  $c$ , we associate  $b^c$  such that  $b_i^c = 0$  if  $c(i) = \text{tgt}$  and  $b_i^c = +\infty$  otherwise: this configuration imposes no constraint, since no edges has been taken yet. Intuitively, in configuration  $(d, b)$ ,  $b$  is used to enforce  $\mu$ -consistency: each edge taken along a path imposes a constraint on the cost of the players for the rest of the path; this constraint is added to the constraints of the earlier edges, and propagated along the path. We can prove that the number of reachable states from  $(c, b^c)$  in  $\mathbb{C}[\mu, c]$ , which we denote with  $|C'|_r$ , is bounded by  $|C| \cdot (n \cdot |V|^n \cdot \kappa)^{|V|}$ .

Computing  $\lambda^{j*}$  from  $\lambda^{(j+1)*}$  amounts to inductively computing  $(\mu_i^{k+1})_{i \in \llbracket n \rrbracket}$  from  $\mu^k$  for edges  $e = (c, w, c') \in Z_j$ , until stabilization. Since  $\mathbb{C}[\mu^k, d]$  can be proved to capture  $\mu^k$ -consistent paths from  $d$ , the computation mainly amounts to checking the existence of paths in such counter graphs, which can be performed in doubly-exponential space. Stabilization can be shown to occur within  $|V|(1 + n \cdot \kappa \cdot |E|^n)$  steps. In the end:

► **Theorem 19.** *The existence of SPEs in a dynamic NCG can be decided in 2EXPSpace.*

► **Remark 20.** Again, our algorithm is not specific to the symmetric setting of our dynamic NCGs; in an asymmetric context, where the number of players would be given in unary, our algorithm would run in EXPSpace.

**Existence of constrained SPEs.** The algorithm above can be extended to compute the cost of the best and worst SPEs, and to include constraints on the costs (both social and individual) of the SPEs we are looking for.

First, for any vector  $\vec{\gamma} = (\gamma_i)_{i \in \llbracket n \rrbracket}$ , we define the  $\vec{\gamma}$ -counter graph  $\mathbb{C}[\lambda^*, c_{\text{src}}, \vec{\gamma}]$ , which is obtained from  $\mathbb{C}[\lambda^*, c_{\text{src}}]$  by replacing the cost vector  $w$  on the edges with  $\vec{\gamma} \cdot w$ .

We can then compute the cost of a  $\vec{\gamma}$ -minimal SPE by checking existence of a path from  $(c_{\text{src}}, b^{c_{\text{src}}})$  to  $(c_{\text{tgt}}, b)$  in  $\mathbb{C}[\lambda^*, c_{\text{src}}, \vec{\gamma}]$ , which minimizes the  $\vec{\gamma}$ -weighted social cost. Again, letting  $\gamma_i = 1$  for all  $i \in \llbracket n \rrbracket$ , a  $\vec{\gamma}$ -minimal SPE is a best SPE, while taking  $\gamma_i = -1$  for all  $i \in \llbracket n \rrbracket$ , we get a worst SPE (maximizing social cost).

We can also solve the constrained-SPE-existence problem by non-deterministically guessing an outcome and checking that it is a path in  $\mathbb{C}[\lambda^{0^*}, c_{src}]$  and that it satisfies the constraints. In each case, we can inductively build a strategy profile witnessing the fact that the selected path is the outcome of an SPE.

## 6 Conclusion and future works

In this paper, we introduced dynamic network congestion games, and studied the complexity of various decision and computation problems concerning social optima, Nash equilibria and subgame perfect equilibria. Our algorithms allow us to compute the price of anarchy and price of stability for those games.

There are couple of areas that are left open in our discussion: possibly the foremost one being the complexity gaps of the decision problems we talked about. As of yet, we do not have interesting lower bounds for constraint NE or constraint SPE problem, so definitely one direction is there for completing the picture. Another aspect of what we do not address in this paper is to obtain *bounds* on PoA/PoSs of our model. Even though we are specifically interested in the measure(s) for a given instance, nonetheless obtaining such bounds could be interesting.

What we are mostly interested in as future work, is to compute how the price of anarchy and the price of stability (and costs of equilibria and social optimum) evolve when the number of players, seen as a parameter, grows.

---

## References

- 1 Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008. doi:10.1137/070680096.
- 2 Guy Avni, Shibashis Guha, and Orna Kupferman. Timed network games. In Kim Guldstrand Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS'17)*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 37:1–37:16. Leibniz-Zentrum für Informatik, August 2017. doi:10.4230/LIPIcs.MFCS.2017.37.
- 3 Guy Avni, Shibashis Guha, and Orna Kupferman. Timed network games with clocks. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS'18)*, volume 117 of *Leibniz International Proceedings in Informatics*, pages 23:1–23:18. Leibniz-Zentrum für Informatik, August 2018. doi:10.4230/LIPIcs.MFCS.2018.23.
- 4 Guy Avni, Thomas A. Henzinger, and Orna Kupferman. Dynamic resource allocation games. *Theoretical Computer Science*, 807:42–55, February 2020. doi:10.1016/j.tcs.2019.06.031.
- 5 Baruch Awerbuch, Yossi Azar, and Amir Epstein. Large the price of routing unsplittable flow. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on the Theory of Computing (STOC'05)*, pages 57–66. ACM Press, May 2005. doi:10.1145/1060590.1060599.
- 6 Nathalie Bertrand, Nicolas Markey, Suman Sadhukhan, and Ocan Sankur. Dynamic network congestion games. *CoRR*, abs/2009.13632, 2020. arXiv:2009.13632.
- 7 Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice Hall, 1989.
- 8 Umang Bhaskar, Lisa Fleischer, and Elliot Anshelevich. A Stackelberg strategy for routing flow over time. *Games and Economic Behavior*, 92:232–247, July 2015. doi:10.1016/j.geb.2013.09.004.

- 9 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie Van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In Wan J. Fokkink and Rob van Glabbeek, editors, *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR'19)*, volume 140 of *Leibniz International Proceedings in Informatics*, pages 13:1–13:16. Leibniz-Zentrum für Informatik, August 2019. doi:10.4230/LIPIcs.CONCUR.2019.13.
- 10 Martin Hoefer, Vahab S. Mirrokni, Heiko Röglin, and Shang-Hua Teng. Competitive routing over time. *Theoretical Computer Science*, 412(39):5420–5432, September 2011. doi:10.1016/j.tcs.2011.05.055.
- 11 Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Gabor Rudolf, and Jihui Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43(2):204–233, August 2008. doi:10.1007/s00224-007-9025-6.
- 12 Ronald Koch and Martin Skutella. Nash equilibria and the price of anarchy for flows over time. *Theory of Computing Systems*, 49(1):71–97, July 2011. doi:10.1007/s00224-010-9299-y.
- 13 Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, May 2009. doi:10.1016/j.cosrev.2009.04.003.
- 14 Elias Koutsoupias and Katia Papakonstantinou. Contention issues in congestion games. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12) – Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 623–635. Springer-Verlag, July 2012. doi:10.1007/978-3-642-31585-5\_55.
- 15 Carol A. Meyers and Andreas S. Schulz. The complexity of welfare maximization in congestion games. *Networks*, 59(2):252–260, March 2012. doi:10.1002/net.20439.
- 16 Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, May 1996. doi:10.1006/game.1996.0044.
- 17 Michal Penn, Maria Polukarov, and Moshe Tennenholtz. Random order congestion games. *Mathematics of Operations Research*, 34(3):706–725, August 2009. doi:10.1287/moor.1090.0394.
- 18 Lili Qiu, Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in internet-like environments. *IEEE Transactions on Computers*, 14(4):725–738, August 2006. doi:10.1109/TNET.2006.880179.
- 19 Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, December 1973. doi:10.1007/BF01737559.
- 20 Tim Roughgarden. Routing games. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 18, page 461–486. Cambridge University Press, 2007.
- 21 Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. *Algorithmica*, 47(1):79–96, 2007. doi:10.1007/s00453-006-1211-4.

# On the Succinctness of Alternating Parity Good-For-Games Automata

Udi Boker

Interdisciplinary Center (IDC) Herzliya, Israel  
udiboker@gmail.com

Denis Kuperberg 

CNRS, LIP, École Normale Supérieure, Lyon, France  
denis.kuperberg@ens-lyon.fr

Karoliina Lehtinen 

University of Liverpool, UK  
k.lehtinen@liverpool.ac.uk

Michał Skrzypczak 

Institute of Informatics, University of Warsaw, Poland  
mskrzypczak@mimuw.edu.pl

---

## Abstract

We study alternating parity good-for-games (GFG) automata, i.e., alternating parity automata where both conjunctive and disjunctive choices can be resolved in an online manner, without knowledge of the suffix of the input word still to be read.

We show that they can be exponentially more succinct than both their nondeterministic and universal counterparts. Furthermore, we present a single exponential determinisation procedure and an EXPTIME upper bound to the problem of recognising whether an alternating automaton is GFG.

We also study the complexity of deciding “half-GFGness”, a property specific to alternating automata that only requires nondeterministic choices to be resolved in an online manner. We show that this problem is PSPACE-hard already for alternating automata on finite words.

**2012 ACM Subject Classification** Theory of computation → Logic and verification

**Keywords and phrases** Good for games, history-determinism, alternation

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.41

**Related Version** A full version is available at <https://arxiv.org/abs/2009.14437>.

**Funding** *Udi Boker*: Israel Science Foundation grant 1373/16.

*Karoliina Lehtinen*: This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 892704.

*Michał Skrzypczak*: Supported by Polish National Science Centre grant no. 2016/21/D/ST6/00491.

## 1 Introduction

*Good-for-games* (GFG) automata were first introduced in [12] as a tool for solving the synthesis problem. The equivalent notion of *history-determinism* was introduced independently in [8] in the context of regular cost functions. Intuitively, a nondeterministic automaton is GFG if nondeterminism can be resolved on the fly, only with knowledge of the input word read so far. GFG automata can be seen as an intermediate formalism between deterministic and nondeterministic ones, with advantages from both worlds. Indeed, like deterministic automata, GFG automata enjoy good compositional properties – useful for solving games and composing automata and trees – and easy inclusion checks [3]. Like nondeterministic automata, they can be exponentially more succinct than deterministic automata [16].



© Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 41; pp. 41:1–41:13



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In recent years, much effort has gone into understanding various properties of nondeterministic GFG automata, for instance their relationship with deterministic automata [3, 16, 5, 15], applications in probabilistic model checking [14] and synthesis of LTL,  $\mu$ -calculus and context-free properties [13, 18], decision procedures for GFGness [19, 16, 2], minimisation [1], and links with recent advances in parity games [10].

Alternating GFG automata are a natural generalisation of nondeterministic GFG automata that enjoy the same compositional properties as nondeterministic GFG automata, while providing more flexibility. As we show in the present work, for some languages alternating GFG parity automata can also be exponentially more succinct, allowing for better synthesis procedures. Indeed, two-player games with winning conditions given by alternating GFG automata are solvable in quasipolynomial time, via a linear reduction to parity games, while for winning conditions given by arbitrary alternating automata, solving games requires determinisation and has therefore double-exponential complexity.

Alternating GFG automata were introduced independently by Colcombet [9] and Quir1 [21] while a form of alternating GFG automata with requirements specific to counters were also considered in [17], as a tool to study cost functions on infinite trees. Boker and Lehtinen studied the expressiveness and succinctness of alternating GFG automata in [6], showing that they

- are not more succinct than DFAs on finite words,
- are as expressive as deterministic ones of the same acceptance condition on infinite words,
- and can be determined with a  $2^{\theta(n)}$  size blowup for the Büchi and coBüchi conditions.

Many questions about GFG alternating automata were left open, in particular whether there exists a doubly exponential succinctness gap between alternating GFG and deterministic automata, and the complexity of deciding whether an alternating parity automaton is GFG.

**Succinctness of alternating GFG automata.** We show that there is a single exponential gap between alternating parity GFG automata and deterministic ones, thereby answering a question left open in [6]. This is in contrast to general alternating automata, for which determinisation incurs a double-exponential size increase. However, we also show that alternating GFG automata can present exponential succinctness compared to both nondeterministic and universal GFG automata. This means that alternating GFG automata can be used to reduce the complexity of solving some games with complex acceptance conditions.

**Recognising GFG automata.** We give an EXPTIME upper bound to the problem of deciding whether an alternating parity automaton is GFG, matching the known upper bound for recognising nondeterministic parity GFG automata.

We also study the complexity of deciding “half-GFGness”, i.e., whether the nondeterminism (or universality) of an automaton is GFG. This property guarantees that composition with games preserves the winner for one of the players. We show that already on finite words, this problem is PSPACE-hard, and it is in EXPTIME for alternating Büchi automata. This shows that a PTIME algorithm for deciding GFGness must exploit the subtle interplay between nondeterminism and universality, and cannot be reduced to checking independently whether each of them is GFG.

## Roadmap

We begin with some definitions, after which, in Section 3, we define alternating GFG automata, study their succinctness and the complexity of deciding half-GFGness, that is, whether the nondeterminism within an alternating automaton is GFG. Section 4 provides a

single-exponential determinisation procedure for alternating GFG parity automata. Section 5 shows that GFGness of alternating parity automata is in EXPTIME, using the determinisation of the previous section. Throughout the paper, we provide high-level proof sketches, with detailed technical developments in the full version [4].

## 2 Preliminaries

**Words and automata.** An *alphabet*  $\Sigma$  is a finite nonempty set of letters. A finite (resp. infinite) *word*  $u = u_0 \dots u_k \in \Sigma^*$  (resp.  $w = w_0 w_1 \dots \in \Sigma^\omega$ ) is a finite (resp. infinite) sequence of letters from  $\Sigma$ . A *language* is a set of words, and the empty word is written  $\epsilon$ . We denote a set  $\{i, i+1, \dots, j\}$  of integers by  $[i, j]$ .

An *alternating word automaton* is a tuple  $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ , where:  $\Sigma$  is an alphabet;  $Q$  is a finite nonempty set of states;  $\iota \in Q$  is an initial state;  $\delta: Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$  is a transition function where  $\mathcal{B}^+(Q)$  is the set of positive Boolean formulas (*transition conditions*) over  $Q$ ; and  $\alpha$ , on which we elaborate below, is either an acceptance condition or a transition labelling on top of which an acceptance condition is defined. For a state  $q \in Q$ , we denote by  $\mathcal{A}^q$  the automaton that is derived from  $\mathcal{A}$  by setting its initial state  $\iota$  to  $q$ .

An automaton  $\mathcal{A}$  is nondeterministic (resp. universal) if all its transition conditions are disjunctions (resp. conjunctions), and it is deterministic if all its transition conditions are just states. We represent the transition function of nondeterministic and universal automata as  $\delta: Q \times \Sigma \rightarrow 2^Q$ , and of a deterministic automaton as  $\delta: Q \times \Sigma \rightarrow Q$ . A *transition* of an automaton is a triple  $(q, a, q') \in Q \times \Sigma \times Q$ , sometimes also written  $q \xrightarrow{a} q'$ .

We denote by  $\widehat{\delta} \subseteq \mathcal{B}^+(Q)$  the set of all subformulas of formulas in the image of  $\delta$ , i.e., all the Boolean formulas that “appear” somewhere in the transition function of  $\mathcal{A}$ .

**Acceptance conditions.** There are various acceptance (winning) conditions, defined with respect to the set of transitions<sup>1</sup> that a path of  $\mathcal{A}$  visits infinitely often. (Notice that a transition condition allows for many possible transitions.) We later formally define acceptance of a word  $w$  by  $\mathcal{A}$  in terms of games, and consider a path of  $\mathcal{A}$  on a word  $w$  as a play in that game. For nondeterministic automata, a “run” coincides with a “path”.

Some of the acceptance conditions are defined on top of a labelling of the transitions rather than directly on the transitions. In particular, in the parity condition, we have  $\alpha: Q \times \Sigma \times Q \rightarrow \Gamma$ , where  $\Gamma \subseteq \mathbb{N}$  is a finite set of priorities and a path is accepting if and only if the highest priority seen infinitely often on it is even.

The Büchi and coBüchi conditions are special cases of the parity condition with  $\Gamma = \{1, 2\}$  and  $\Gamma = \{0, 1\}$ , respectively. When speaking of Büchi and coBüchi automata, we often refer to  $\alpha$  as the set of “accepting transitions”, namely the transitions that are mapped to 2 in the Büchi case and to 0 in the coBüchi case. The weak condition is a special case of both the Büchi and coBüchi conditions, in which every path eventually remains in the same priority.

The Rabin and Streett conditions are more involved, yet defined directly on the set  $T$  of transitions. A Rabin condition is a set  $\{(B_1, G_1), (B_2, G_2), \dots, (B_k, G_k)\}$ , with  $B_i, G_i \subseteq T$ , and a path  $\rho$  is accepting iff for some  $i \in [1, k]$ , we have that the set  $\text{inf}(\rho)$  of transitions that are visited infinitely often in  $\rho$  satisfies  $(\text{inf}(\rho) \cap B_i = \emptyset \text{ and } \text{inf}(\rho) \cap G_i \neq \emptyset)$ . A Streett condition is dual: a set  $\{(B_1, G_1), (B_2, G_2), \dots, (B_k, G_k)\}$ , with  $B_i, G_i \subseteq Q$ , whereby a path  $\rho$  is accepting iff for all  $i \in [1, k]$ , we have  $(\text{inf}(\rho) \cap B_i = \emptyset \text{ or } \text{inf}(\rho) \cap G_i \neq \emptyset)$ .

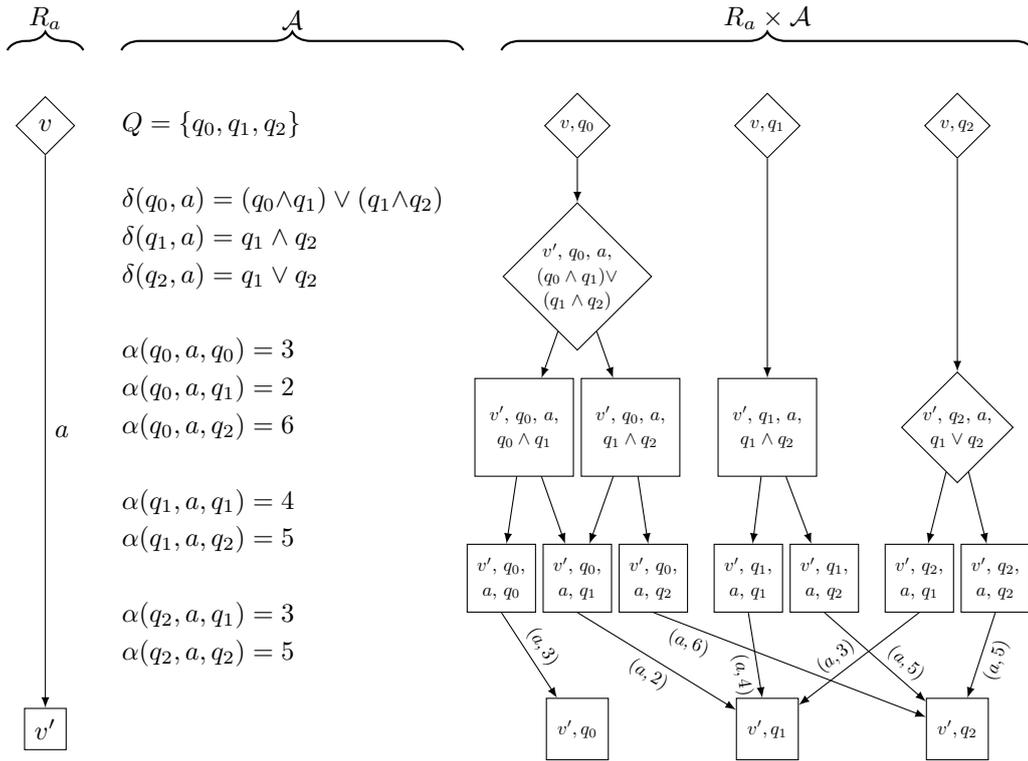
<sup>1</sup> Acceptance is defined in the literature with respect to either states or transitions; for technical reasons we prefer to work with acceptance on transitions.

**Sizes and types of automata.** The size of  $\mathcal{A}$  is the maximum of the alphabet size, the number of states, the transition function length, which is the sum of the transition condition lengths over all states and letters, and the acceptance condition's index, which is 1 for weak, Büchi and coBüchi,  $|\Gamma|$  for parity, and  $k$  for Rabin and Street.

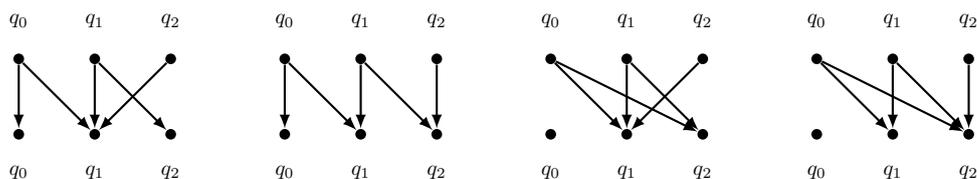
We sometimes abbreviate automata types by three-letter acronyms in  $\{D, N, U, A\} \times \{F, W, B, C, P, R, S\} \times \{A, W\}$ . The first letter stands for the transition mode, the second for the acceptance condition, and the third indicates that the automaton runs on finite or infinite words. For example, DPW stands for a deterministic parity automaton on infinite words.

**Games and strategies.** Some of our technical proofs use standard concepts of an arena, a game, a winning strategy, etc. For the sake of completeness, we provide precise mathematical definitions of these objects in the full version [4]. Here we will just overview the involved concepts.

First, we work with two-player games of perfect information, where the players are Eve and Adam. These games are played on graphs (called arenas). Most of the considered games are of infinite duration and their winning condition is expressed in terms of the infinite sequences of edges taken during the play. We invoke results of determinacy (one of the players has a winning strategy), as well as of *positional determinacy* (one of the players has a strategy that depends only on the last position of the play).



■ **Figure 1** A one-step arena over a letter  $a \in \Sigma$ , obtained as a product of a simple arena  $R_a$  with the alternating parity automaton  $\mathcal{A}$ . In this example  $v$  is controlled by Eve and  $v'$  by Adam. The transitions with no label are labelled by  $\epsilon$ . Diamond-shaped positions belong to Eve and square-shaped positions belong to Adam.



■ **Figure 2** The four possible boxes corresponding to Eve’s choices in the one-step arena of Figure 1. (All edges should be labelled with  $a$ , which we omit for better readability.)

**Model-checking games.** To represent the semantics of an alternating automaton  $\mathcal{A}$ , we treat the Boolean formulas that appear in the transition conditions of  $\mathcal{A}$  as games. More precisely, given a letter  $a \in \Sigma$  we represent the transition conditions  $q \mapsto \delta(q, a) \in \mathbf{B}^+(Q)$  as the *one-step arena* over  $a$  (see Figure 1). A play over this arena begins in a state  $q \in Q$ ; then the players go down the formula  $\delta(q, a)$  with Eve resolving disjunctions and Adam resolving conjunctions; and finally they reach an atom  $q' \in Q$  and the play stops. This means that a play over the one-step arena over  $a$  results in a transition of the form  $q \xrightarrow{a} q'$ .

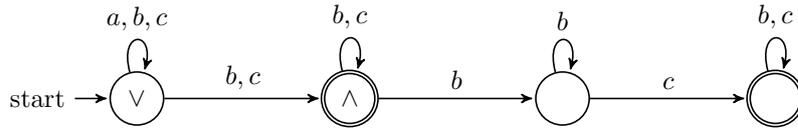
The language  $L(\mathcal{A})$  of an alternating automaton  $\mathcal{A}$  over an alphabet  $\Sigma$  is defined via the *model-checking game*, defined for an automaton  $\mathcal{A}$  and a word  $w = a_0a_1a_2 \dots \in \Sigma^\omega$ . A *configuration* of this game is a state  $q$  of  $\mathcal{A}$  and a position  $i \in \omega$  of  $w$ , starting at  $(\iota, 0)$ . In the  $i$ th round, starting at configuration  $(q_i, i)$ , the players play on the one-step arena from  $q_i$  over  $a_i$ , resulting in a transition  $q_i \xrightarrow{a_i} q_{i+1}$ . The next configuration is  $(q_{i+1}, i+1)$ . The acceptance condition of  $\mathcal{A}$  becomes the winning condition of this game.  $\mathcal{A}$  *accepts*  $w$  if Eve has a winning strategy in this game.

For technical convenience, we define (in the full version) the model-checking game in terms of a *synchronised product* of the word  $w$  (treated as an infinite graph) and the automaton  $\mathcal{A}$ . Synchronised products turn out to be useful in the analysis of various games presented in this paper and are used throughout the technical version of the paper [4].

► **Definition 1.** Given an alternating automaton  $\mathcal{A}$ , we denote by  $\overline{\mathcal{A}}$  the dual automaton: it has the same alphabet, set of states, and initial state. Its transition conditions  $\delta_{\overline{\mathcal{A}}}(q, a)$  are obtained from those of  $\mathcal{A}$  by replacing each disjunction  $\vee$  with conjunction  $\wedge$  and vice versa. Its acceptance condition is the dual of  $\mathcal{A}$ ’s condition. (In parity automata, all priorities are increased by 1.)  $\overline{\mathcal{A}}$  recognises the complement  $L(\mathcal{A})^c$  of  $L(\mathcal{A})$ .

**Boxes.** Another technical concept that we use is that of *boxes* (see Figure 2), which describe Eve’s *local* strategies for resolving disjunctions within a transition condition. Consider an alternating automaton  $\mathcal{A}$  and a letter  $a \in \Sigma$ . Moreover, fix a strategy  $\sigma$  of Eve that resolves disjunctions in all the transition conditions  $\delta(q, a)$  for  $q \in Q$ . Now, the *box* of  $\mathcal{A}$ ,  $a$ , and  $\sigma$  is a subset of  $Q \times \Sigma \times Q$  and contains a triple  $(q, a, q')$  iff  $\sigma$  resolves disjunctions of  $\delta(q, a)$  in such a way that Adam (resolving conjunctions) can reach the atom  $q'$ . In other words, this box contains  $(q, a, q')$  if there is a play consistent with  $\sigma$  on  $\delta(q, a)$  that reaches the atom  $q'$ . We use  $\beta$  to denote single boxes and by  $\mathbf{B}_{\mathcal{A}, a}$  we denote the set of all boxes of  $\mathcal{A}$  and  $a$ , while  $\mathbf{B}_{\mathcal{A}}$  denotes the union  $\bigcup_{a \in \Sigma} \mathbf{B}_{\mathcal{A}, a}$ . We give a more formal definition based on synchronised products in the full version [4].

► **Definition 2.** Given a sequence of boxes  $\pi = b_0, b_1, \dots$  of an automaton  $\mathcal{A}$  and a path  $\rho = (q_0, a_0, q_1), (q_1, a_1, q_2), \dots$ , we say that  $\rho$  is a path of  $\pi$  if for every  $i$  we have  $(q_i, a_i, q_{i+1}) \in b_i$ . The sequence  $\pi$  is said to be *universally accepting* if every path in  $\pi$  is accepting in  $\mathcal{A}$ .



■ **Figure 3** Alternating weak automaton accepting words over  $\{a, b, c\}$  in which  $a$  occurs finitely often and  $c$  occurs infinitely often. Omitted transitions lead to a rejecting sink.

Intuitively, a sequence of boxes  $\pi$  as above represents a particular positional strategy  $\sigma$  of Eve in the model-checking game over the word  $w = a_0a_1a_2\dots$ . In that case, a path of  $\pi$  corresponds to a possible play of this game consistent with  $\sigma$ , and the sequence is universally accepting if and only if the strategy is winning.

### 3 Good-For-Games Alternating Automata

Good-for-games (GFG) nondeterministic automata are automata in which the nondeterministic choices can be resolved without looking at the future of the word. For example, consider an automaton that consists of a nondeterministic choice between a component that accepts words in which  $a$  occurs infinitely often and a component that accepts words in which  $a$  occurs finitely often. This automaton accepts all words but is not GFG since the nondeterministic choice of component cannot be resolved without knowing the whole word.

To extend this definition to alternating automata, we must look both at its nondeterminism and universality and require that both can be resolved without knowledge of the future. The following letter games capture this intuition.

► **Definition 3** (Letter games [6]). *Given an alternating automaton  $\mathcal{A}$ , Eve’s letter game proceeds at each turn from a state  $q$  of  $\mathcal{A}$ , starting from the initial state of  $\mathcal{A}$ , as follows:*

- Adam chooses a letter  $a$ ,
- Adam and Eve play on the one-step arena over  $a$  from  $q$  to a new state  $q'$ , where Eve resolves disjunctions and Adam conjunctions.

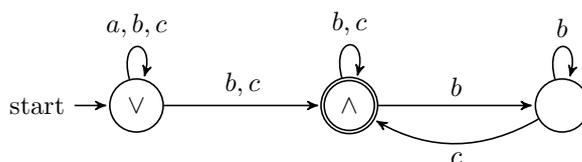
A play of the letter game thus generates a word  $w$  and a path  $\rho$  of  $\mathcal{A}$  on  $w$ . Eve wins this play if either  $w \notin L(\mathcal{A})$  or  $\rho$  is accepting in  $\mathcal{A}$ .

Adam’s letter game is similar, except that Eve chooses letters and Adam wins if either  $w \in L(\mathcal{A})$  or the path  $\rho$  is rejecting.

► **Definition 4** (GFG automata [6]). *An automaton  $\mathcal{A}$  is  $\exists$ -GFG if Eve wins her letter game; it is  $\forall$ -GFG if Adam wins his letter game. Finally,  $\mathcal{A}$  is GFG if it is both  $\exists$ -GFG and  $\forall$ -GFG.*

As shown in [6, Theorem 8], an automaton  $\mathcal{A}$  is GFG if and only if it is indeed “good for playing games”, in the sense that its product with every game whose winning condition is  $L(\mathcal{A})$  preserves the winner of the game.

► **Example 5.** The automaton in Figure 3 accepts the language  $L$  of words in which  $a$  occurs finitely often and  $c$  occurs infinitely often. Here Eve loses her letter game: Adam can play  $c$  until Eve takes the transition to the second state, and then play  $a$  followed by  $c^\omega$ . Conversely, Eve wins Adam’s letter game: her strategy is to play  $b$ , take the transition to the second state and keep playing  $b$  until Adam takes the transition into the third state, after which she plays  $c$  once and then  $b^\omega$ . This automaton is neither  $\exists$ -GFG nor  $\forall$ -GFG, and taking its product with games with  $L$  as winning condition does not preserve the winner of the game.



■ **Figure 4** Alternating coBüchi  $\forall$ -GFG automaton accepting words over  $\{a, b, c\}$  in which  $a$  occurs finitely often and  $c$  occurs infinitely often.

In contrast, the automaton in Figure 4 is  $\forall$ -GFG but not  $\exists$ -GFG. Indeed, Adam’s winning strategy in his letter game is to resolve the conjunction from the middle state by always moving to the right-hand state when Eve plays  $b$ . This forces Eve to choose between playing  $c$  infinitely many times (in which case, the word is in the language) or letting Adam build a rejecting run. Taking its product with *one-player games* with winning condition  $L$  preserves the winner whenever Eve is the player controlling all positions. However, this is not the case for one-player games where Adam is the sole player.

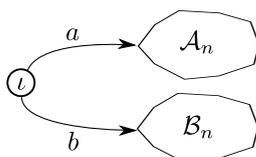
### 3.1 Alternating GFG vs. Nondeterministic and Universal Ones

We show in this section that alternating GFG automata can be more succinct than both nondeterministic and universal GFG automata.

► **Lemma 6.** *There is a family  $(\mathcal{C}_n)_{n \in \mathbb{N}}$  of alternating GFG  $\{0, 1, 2\}$ -parity automata of size linear in  $n$  over a fixed alphabet, such that every nondeterministic GFG parity automaton and universal GFG parity automaton for  $L(\mathcal{C}_n)$  is of size  $2^{\Omega(n)}$ .*

**Proof.** From [16], there is a family  $(\mathcal{A}_n)_{n \in \mathbb{N}}$  of GFG-NCWs with  $n$  states over a fixed alphabet  $\Sigma$ , such that every DPW for  $L_n = L(\mathcal{A}_n)$  is of size  $2^{\Omega(n)}$ . For every  $n \in \mathbb{N}$ , let  $\mathcal{B}_n$  be the dual of  $\mathcal{A}_n$ , so  $\mathcal{B}_n$  is a UBW accepting  $\overline{L_n}$ . We build an APW  $\mathcal{C}_n$  over  $\Sigma$  of size linear in  $n$ , by setting its initial state to move to the initial state of  $\mathcal{A}_n$  when reading the letter  $a \in \Sigma$  and to the initial state of  $\mathcal{B}_n$  when reading the letter  $b \in \Sigma$ . The acceptance condition of  $\mathcal{C}_n$  is a parity condition with priorities  $\{0, 1, 2\}$ : accepting transitions of  $\mathcal{A}_n$  are assigned priority 0, and accepting transitions of  $\mathcal{B}_n$  priority 2. Other transitions have priority 1.

The automaton  $\mathcal{C}_n$  is represented below:



Observe that  $L(\mathcal{C}_n) = aL_n \cup b\overline{L_n}$ , and that  $\mathcal{C}_n$  is GFG: its initial state has only deterministic transitions, and over the  $\mathcal{A}_n$  and  $\mathcal{B}_n$  components, the strategy to resolve the nondeterminism and universality, respectively, follows the strategy to resolve the nondeterminism of  $\mathcal{A}_n$ , which is guaranteed due to  $\mathcal{A}_n$ ’s GFGness.

Consider a GFG UPW  $\mathcal{E}_n$  for  $L(\mathcal{C}_n)$ , and let  $q$  be a state to which  $\mathcal{E}_n$  moves when reading  $a$ , according to some strategy that witnesses  $\mathcal{E}_n$ ’s GFGness. Then  $\mathcal{E}_n^q$  is a GFG UPW for  $L_n$ . Its dual is therefore a GFG NPW  $\mathcal{E}'_n$  for  $\overline{L_n}$ .

Since  $\mathcal{A}_n$  is a GFG NPW for  $L_n$ , by [3, Theorem 4] we obtain a DPW for  $L_n$  of size  $|\mathcal{A}_n||\mathcal{E}'_n|$ . By choice of  $L_n$ , this DPW must be of size  $2^{\Omega(n)}$ , and since  $\mathcal{A}_n$  is of size  $n$ , it follows that  $\mathcal{E}'_n$ , and hence  $\mathcal{E}_n$ , must be of size  $2^{\Omega(n)}$ . By a symmetric argument, every GFG NPW for  $L(\mathcal{C}_n)$  must also be of size  $2^{\Omega(n)}$ . ◀

Informally, the language  $L_n$  above describes a set of threads, of which at least one eventually satisfies a safety property. Then, the above construction can be understood as describing a property of reactive systems where, depending on the input, the system guarantees either that there is a thread that eventually satisfies a safety property, or that all threads satisfy a liveness (Büchi) property. The GFG alternating automaton can then be used to solve in polynomial time games with such languages as winning condition, for example in the context of synthesis: the product of the game arena and the alternating automaton for  $L_n$  is a parity game with 3 priorities with the same winner as the original game. In contrast, a DPW, GFG NPW and GFG UPW for the same language would all be exponentially larger.

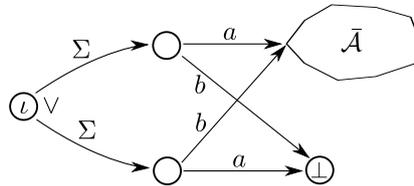
### 3.2 Deciding Half-GFGness

In order to decide GFGness, it is enough to be able to decide the  $\exists$ -GFG property on the automaton and its dual. A natural first approach is therefore to study the complexity of deciding whether an APW is  $\exists$ -GFG. Yet, we will show that already on finite words, this problem is PSPACE-hard, while we conjecture that deciding GFGness is in PTIME.

► **Lemma 7.** *Deciding whether an AFA is  $\exists$ -GFG is PSPACE-hard.*

**Proof.** We reduce from NFA universality: starting from an NFA  $\mathcal{A}$ , we build an AFA  $\mathcal{B}$  based on the dual of  $\mathcal{A}$ , with an additional non-GFG choice to be resolved by Eve. This AFA  $\mathcal{B}$  is  $\exists$ -GFG if and only if  $L(\mathcal{B}) = \emptyset$ , which happens if and only if  $L(\mathcal{A}) = \Sigma^*$ . We crucially use the fact that  $\mathcal{B}$  is not necessarily  $\forall$ -GFG.

Let  $\mathcal{A}$  be an NFA over an alphabet  $\Sigma = \{a, b\}$  and  $\bar{\mathcal{A}}$  its dual. We want to check whether  $L(\mathcal{A}) = \Sigma^*$ . We build an AFA  $\mathcal{B}$ , as depicted below, by first making Eve guess the second letter. If her guess is wrong, the automaton proceeds to a rejecting sink state  $\perp$ . Otherwise, it proceeds to the initial state of  $\bar{\mathcal{A}}$ . The size of  $\mathcal{B}$  is linear in the size of  $\mathcal{A}$ .



If  $L(\bar{\mathcal{A}}) = \emptyset$ , then  $L(\mathcal{B}) = \emptyset$ , so  $\mathcal{B}$  is trivially  $\exists$ -GFG. However, if there is some  $u \in L(\bar{\mathcal{A}})$ , then Adam has a winning strategy in Eve's letter game on  $\mathcal{B}$ . This strategy consists of playing  $a$ , then playing the letter that brings Eve to  $\perp$ , and finally playing  $u$ . The resulting word is in  $L(\mathcal{B}) = \Sigma^2 L(\bar{\mathcal{A}})$ , so this witnesses that  $\mathcal{B}$  is not  $\exists$ -GFG. We obtain that  $L(\mathcal{A}) = \Sigma^* \Leftrightarrow L(\bar{\mathcal{A}}) = \emptyset \Leftrightarrow \mathcal{B}$  is  $\exists$ -GFG, which is the wanted reduction. ◀

For Büchi automata, and so in particular for finite words, we can give an EXPTIME algorithm for this problem.

► **Lemma 8.** *Deciding whether an ABW is  $\exists$ -GFG is in EXPTIME.*

**Proof.** It is shown in [6, Lemma 23] that removing alternation from an ABW  $\mathcal{A}$  using the breakpoint construction [20] yields an NBW  $\mathcal{B}$  such that if  $\mathcal{A}$  is  $\exists$ -GFG then  $\mathcal{B}$  is GFG. Moreover, the converse also holds: if  $\mathcal{B}$  is GFG then  $\mathcal{A}$  is  $\exists$ -GFG, since playing Eve's letter game in  $\mathcal{B}$  is more difficult for Eve than playing it in  $\mathcal{A}$ . This means that starting from an ABW  $\mathcal{A}$ , we can build an exponential size NBW  $\mathcal{B}$  via breakpoint construction, and test whether  $\mathcal{B}$  is GFG via the algorithm from [2], in time polynomial with respect to  $\mathcal{B}$ . Overall, this yields an EXPTIME algorithm deciding whether  $\mathcal{A}$  is  $\exists$ -GFG. ◀

## 4 Determinisation of Alternating GFG Parity Automata

In this section we provide a procedure that, given an alternating GFG parity automaton, produces an equivalent deterministic parity automaton with single-exponentially many states. To do so, we first provide an alternation-removal procedure that preserves GFG status. Then, we apply this procedure to both the input automaton and its complement and use the GFG strategies in these two automata to determinise the input automaton. Our proofs, in [4] rely on some analysis of when GFG strategies can use the history of the word, rather than the full history of the play (which also includes the choices of how to resolve the nondeterminism and universality), and on the memoryless determinacy of parity games.

Our method for going from alternating to nondeterministic automata is similar to that of Dax and Klaedtke [11]: they take a nondeterministic automaton that recognises the universally-accepting words in  $(B_A)^\omega$  and add nondeterminism that upon reading a letter  $a \in \Sigma$  chooses a box in  $B_A$  over  $a$ . Yet in our approach, in order to guarantee that the outcome preserves GFGness, the intermediate automaton is deterministic.

### 4.1 Alternation Removal in GFG Parity Automata

► **Theorem 9.** *Consider an alternating parity automaton  $\mathcal{A}$  with  $n$  states and index  $k$ . There exists a nondeterministic parity automaton  $\text{box}(\mathcal{A})$  with  $2^{O(nk \log nk)}$  states that is equivalent to  $\mathcal{A}$  such that if  $\mathcal{A}$  is GFG then  $\text{box}(\mathcal{A})$  is also GFG.*

In Section 5, where we discuss decision procedures, we will show that  $\text{box}(\mathcal{A})$  is GFG *exactly* when  $\mathcal{A}$  is  $\exists$ -GFG. For now, the rest of this section is devoted to the proof of Theorem 9, of which a detailed version can be found in the full version [4].

► **Lemma 10.** *Consider an alternating parity automaton  $\mathcal{A}$  with  $n$  states and index  $k$ . Then there exists a deterministic parity automaton  $\mathcal{B}$  with  $2^{O(nk \log nk)}$  states over the alphabet  $B_A$  that recognises the set of universally-accepting words for  $\mathcal{A}$ . If  $\mathcal{A}$  is a Büchi automaton, then  $\mathcal{B}$  can also be taken as Büchi, and in general the parity index of the automaton  $\mathcal{B}$  is linear in the number of transitions of  $\mathcal{A}$ .*

**Proof sketch.** We first construct a nondeterministic parity (resp. coBüchi) automaton over the alphabet  $B_A$  that recognises the complement of the set of universally-accepting words for  $\mathcal{A}$ . This automaton is easy to build: it guesses a path that is not accepting, and has the dual acceptance condition to  $\mathcal{A}$ . We then obtain the automaton  $\mathcal{B}$  by determinising and complementing this automaton. ◀

We now build the automaton  $\text{box}(\mathcal{A})$  of Theorem 9. It is the same as the automaton  $\mathcal{B}$  of Lemma 10, except that the alphabet is  $\Sigma$  and the transition function is defined as follows: For every state  $p$  of  $\mathcal{B}$  and  $a \in \Sigma$ , we have  $\delta_{\text{box}(\mathcal{A})}(p, a) := \bigcup_{\beta \in B_{(\mathcal{A}, a)}} \delta_{\mathcal{B}}(p, \beta)$ .

In other words, the automaton  $\text{box}(\mathcal{A})$  reads a letter  $a$ , nondeterministically guesses a box  $\beta \in B_{\mathcal{A}, a}$ , and follows the transition of  $\mathcal{B}$  over  $\beta$ . Thus, the runs of  $\text{box}(\mathcal{A})$  over a word  $w = w_0 w_1 w_2 \dots \in \Sigma^\omega$  are in bijection with sequences of boxes  $(\beta_i)_{i \in \mathbb{N}}$  such that  $\beta_i \in B_{\mathcal{A}, w_i}$  for all  $i \in \mathbb{N}$ .

Fix an infinite word  $w \in \Sigma^\omega$ . Our aim is to prove that  $w \in L(\mathcal{A}) \Leftrightarrow w \in L(\text{box}(\mathcal{A}))$ .

► **Lemma 11.** *There exists a bijection between positional strategies of Eve in the acceptance game of  $\mathcal{A}$  over  $w$  and runs of  $\text{box}(\mathcal{A})$  over  $w$ . Moreover, a strategy is winning if and only if the corresponding run is accepting. Thus  $L(\mathcal{A}) = L(\text{box}(\mathcal{A}))$ .*

## 41:10 On the Succinctness of Alternating Parity Good-For-Games Automata

► **Remark 12.** The above alternation-removal procedure also extends to alternating Rabin automata but fails for alternating Streett automata  $\mathcal{A}$ : since Streett games are not positionally determined for Eve, the acceptance game of  $\mathcal{A}$  over a word  $w$  is not positionally determined for Eve.

► **Lemma 13.** *For an alternating  $\exists$ -GFG parity automaton  $\mathcal{A}$ , the automaton  $\text{box}(\mathcal{A})$  is GFG.*

Intuitively, this is because the construction of  $\text{box}(\mathcal{A})$  preserves the nondeterminism of  $\mathcal{A}$ .

### 4.2 Single-Exponential Determinisation

The aim of this section is to prove the following determinisation theorem.

► **Theorem 14.** *If  $\mathcal{A}$  is an alternating parity GFG automaton then there exists a deterministic parity automaton  $\mathcal{D}$  that recognises the same language and has size at most exponential in the size of  $\mathcal{A}$ . Moreover, the parity index of  $\mathcal{D}$  is the same as that of  $\mathcal{A}$ .*

► **Remark 15.** Theorem 9 and [3, Theorem 4], which uses an NRW-GFG and its complement NRW-GFG to obtain a DRW, together give an exponential deterministic parity automaton for  $L(\mathcal{A})$ . However, the index of  $\mathcal{A}$  might not be preserved. On the other hand, from [6, Theorem 19] we know that there exists a deterministic parity automaton equivalent to  $\mathcal{A}$  with the same index, but it might have more than exponentially many states. Here we are able to guarantee both the preservation of the index and an exponential upper bound on the size of the deterministic automaton.

Observe that Theorem 9 can be applied both to  $\mathcal{A}$  and its dual. Therefore, we can fix a pair of nondeterministic GFG parity automata  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$  that recognise  $L(\mathcal{A})$  and  $L(\mathcal{A})^c$  respectively and are both of size exponential in  $\mathcal{A}$ . We use the automata  $\mathcal{A}$ ,  $\text{box}(\mathcal{A})$ , and  $\text{box}(\bar{\mathcal{A}})$  to construct two auxiliary games  $G(\mathcal{A})$  and  $G'(\mathcal{A})$ .

The game  $G(\mathcal{A})$  proceeds from a configuration consisting of a pair  $(p, q)$  of states from  $\text{box}(\bar{\mathcal{A}})$  and  $\mathcal{A}$  respectively, starting from their initial states, as follows:

- Adam chooses a letter  $a \in \Sigma$ ;
- Eve chooses a transition  $p \xrightarrow{a} p'$  in  $\text{box}(\bar{\mathcal{A}})$ ;
- Eve and Adam play on the one-step arena over  $a$  from  $q$  to a new state  $q'$ .

A play in  $G(\mathcal{A})$  consists of a run  $\rho$  in  $\text{box}(\bar{\mathcal{A}})$  and a path  $\rho'$  in  $\mathcal{A}$ . It is winning for Eve if either  $\rho$  is accepting in  $\text{box}(\bar{\mathcal{A}})$  (in which case  $w \notin L(\mathcal{A})$ ), or  $\rho'$  is accepting in  $\mathcal{A}$ .

If  $\mathcal{A}$  is  $\exists$ -GFG and  $\text{box}(\bar{\mathcal{A}})$  is GFG, Eve has a winning strategy in  $G(\mathcal{A})$  consisting of building a run in  $\text{box}(\bar{\mathcal{A}})$  using her GFG strategy in  $\text{box}(\bar{\mathcal{A}})$  and a path in  $\mathcal{A}$  using her  $\exists$ -GFG strategy in  $\mathcal{A}$ . This guarantees that if  $w \in L(\mathcal{A})$  then the path in  $\mathcal{A}$  is accepting, and otherwise the run in  $\text{box}(\bar{\mathcal{A}})$  is accepting.

We then argue that as the winning condition of  $G(\mathcal{A})$  is a Rabin condition, Eve also has a winning strategy that is positional in  $\mathcal{A}$ , that is, which only depends on the history of the word and the current position. (Interestingly, the question of whether Eve can resolve the nondeterminism in a class of alternating GFG automata with only the knowledge of the word read so far does not tightly correspond to whether the acceptance condition of this class is memoryless. For example, it does hold for the generalised-Büchi condition, though it is not memoryless.)

► **Remark 16.** There is some magic here: both the GFG strategies of Eve in  $\mathcal{A}$  and in  $\text{box}(\bar{\mathcal{A}})$  may require exponential memory, yet, when she needs to satisfy the disjunction of the two conditions, no more memory is needed. In a sense, the states of  $\mathcal{A}$  provide the memory for  $\text{box}(\bar{\mathcal{A}})$  and the states of  $\text{box}(\bar{\mathcal{A}})$  provide the memory for  $\mathcal{A}$ .

The game  $G'(\mathcal{A})$  is similar, except that Adam is given control of  $\text{box}(\mathcal{A})$  and Eve is in charge of letters. This time Adam wins a play, consisting of a run of  $\text{box}(\mathcal{A})$  and a path in  $\mathcal{A}$ , if either the path of  $\mathcal{A}$  is rejecting or the run of  $\text{box}(\mathcal{A})$  is accepting.

Accordingly, if  $\mathcal{A}$  is GFG, then he can win by using the  $\exists$ -GFG strategy in  $\text{box}(\mathcal{A})$  and the  $\forall$ -GFG strategy in  $\mathcal{A}$ . Then if  $w \in L(\mathcal{A})$ , the run in  $\text{box}(\mathcal{A})$  is accepting, and otherwise the path of  $\mathcal{A}$  is rejecting. As before, he also has a positional winning strategy in  $G'(\mathcal{A})$ .

We are now ready to build the deterministic automaton from a GFG APW  $\mathcal{A}$ , using positional winning strategies  $\sigma$  and  $\tau$  for Eve and Adam in  $G(\mathcal{A})$  and  $G'(\mathcal{A})$ , respectively.

Let  $\mathcal{D}$  be the automaton with states of the form  $(q, p_1, p_2)$ , with  $q$  a state of  $\mathcal{A}$ ,  $p_1$  a state of  $\text{box}(\mathcal{A})$  and  $p_2$  a state of  $\text{box}(\bar{\mathcal{A}})$ . A transition of  $\mathcal{D}$  over  $a$  moves to  $(q', p'_1, p'_2)$  such that moving from  $(q, p_1)$  to  $(q', p_1)$  is consistent with  $\tau$ ; and moving from  $(q, p_2)$  to  $(q', p'_2)$  is consistent with  $\sigma$ . The acceptance condition of  $\mathcal{D}$  is inherited from  $\mathcal{A}$ .

► **Lemma 17.** *For a GFG APW  $\mathcal{A}$  and  $\mathcal{D}$  built as above,  $L(\mathcal{A}) = L(\mathcal{D})$ .*

► **Remark 18.** To extend this construction to an alternating GFG Rabin automaton  $\mathcal{A}$ , we would need to remove alternations from both  $\mathcal{A}$  and its dual while preserving GFGness. However, the dual is a Streett automaton, for which we cannot invoke positional determinacy.

## 5 Deciding GFGness of Alternating Automata

We use the development of the last section to show that deciding whether an APW is GFG is in EXPTIME. This matches the best known upper bound for the same problem on NPW.

The main result of this section is the following theorem.

► **Theorem 19.** *There exists an EXPTIME algorithm that takes as input an alternating parity automaton  $\mathcal{A}$  and decides whether  $\mathcal{A}$  is GFG.*

The idea is to construct the (exponential size) NPWs  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$  for  $L(\mathcal{A})$  and  $L(\mathcal{A})^c$  respectively, which are GFG if and only if  $\mathcal{A}$  is  $\exists$ -GFG and  $\forall$ -GFG respectively. Then, it remains to check whether both are indeed GFG. Since we don't have a polynomial procedure to check this, instead, we will build a game which Eve wins if and only if *both* are indeed GFG, and which we can solve in exponential time with respect to the size of  $\mathcal{A}$ .

First, we observe the following reciprocal of Lemma 13.

► **Lemma 20.** *If  $\text{box}(\mathcal{A})$  is GFG then  $\mathcal{A}$  is  $\exists$ -GFG.*

**Proof.** Assume that  $\text{box}(\mathcal{A})$  is GFG and consider a strategy witnessing this. Such a strategy can be easily turned into a function  $\sigma': \Sigma^+ \rightarrow \mathcal{B}_{\mathcal{A}}$  that, given a word  $w \in L(\mathcal{A})$  produces a universally accepting word of boxes of  $\mathcal{A}$ . Now, due to the definition of a box, each such box defines a positional strategy of Eve in the respective one-step game. This allows us to construct a winning strategy of Eve in the letter game over  $\mathcal{A}$ . ◀

Thus,  $\mathcal{A}$  is GFG if and only if both  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$  are GFG. To decide this, we consider a game  $G''$  where Adam plays letters and Eve produces runs of the automata  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$  in parallel. The winning condition of  $G''$  requires that at least one of the constructed runs must be accepting.

Now, each sequence of letters given by Adam belongs either to the language of  $\text{box}(\mathcal{A})$  or to  $\text{box}(\bar{\mathcal{A}})$  and therefore, a winning strategy of Eve in  $G''$  must comprise of two strategies witnessing GFGness of both  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$ . Dually, if both  $\text{box}(\mathcal{A})$  and  $\text{box}(\bar{\mathcal{A}})$  are GFG then Eve wins  $G''$  by playing the two strategies in parallel.

It remains to show that  $G''$  is solvable in EXPTIME. Its winning condition is a disjunction of parity conditions, with index linear in the number of transitions of  $\mathcal{A}$ . This winning condition is recognised by a deterministic parity automaton of exponential size with polynomial index. To solve  $G''$ , we take its product with this deterministic automaton that recognises its winning condition, and solve the resulting parity game with an algorithm that is polynomial in the size of the game whenever, like here, the number of priorities is logarithmic in the size of the game, for instance [7]. Details of this construction and its complexity are in the full version [4].

## 6 Conclusions

The results obtained in this work shed new light on where alternating GFG automata resemble nondeterministic ones, and where they differ. Overall, our results show that allowing GFG alternations add succinctness without significantly increasing the complexity of determinisation nor decision procedures.

In particular, we show that alternating parity GFG automata can be exponentially more succinct than any equivalent nondeterministic GFG automata, yet this succinctness does not become double exponential when compared to deterministic automata, answering a question from [6]. Some further succinctness problems are left open here, such as the possibility of a doubly exponential gap between alternating GFG automata of stronger acceptance conditions and deterministic ones, as well as between  $\exists$ -GFG parity automata and deterministic ones.

We also show that the interplay between the two players can be used to decide whether an automaton is GFG without deciding  $\exists$ -GFG and  $\forall$ -GFG separately, yielding an EXPTIME algorithm. This matches the current algorithms for deciding GFGness on non-deterministic automata. Bagnol and Kuperberg conjectured that GFGness is PTIME decidable for non-deterministic parity automata of fixed index [2]; we extend this conjecture to alternating automata.

It then becomes interesting to ask how to build an alternating automaton GFG. Indeed, Henzinger and Piterman [12] proposed a transformation of nondeterministic automata into GFG automata, which, despite in some cases leading to a deterministic automaton, is, conceptually, a much simpler procedure than determinisation. Indeed, in many examples of non-GFG automata, adding transitions suffices to obtain a GFG one. We leave finding such a procedure for alternating automata as future work.

---

## References

- 1 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In *Proceedings of ICALP*, pages 100:1–100:16, 2019.
- 2 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *Proceedings of FSTTCS*, pages 16:1–16:14, 2018.
- 3 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proceedings of ICALP*, pages 89–100, 2013.
- 4 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On the succinctness of alternating parity good-for-games automata, 2020. [arXiv:2009.14437](https://arxiv.org/abs/2009.14437).
- 5 Udi Boker, Orna Kupferman, and Michał Skrzypczak. How deterministic are good-for-games automata? In *Proceedings of FSTTCS*, pages 18:1–18:14, 2017.
- 6 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proceedings of CONCUR*, 2019.
- 7 Cristian S Calude, Sanjay Jain, Bakhadyr Khossainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of STOC*, pages 252–263, 2017.

- 8 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*, pages 139–150, 2009.
- 9 Thomas Colcombet. Fonctions régulières de coût. Habilitation thesis, 2013.
- 10 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Proceedings of FOSSACS*, pages 1–26, 2019.
- 11 Christian Dax and Felix Klaedtke. Alternation elimination by complementation. In *Proceedings of LPAR*, pages 214–229, 2008.
- 12 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.
- 13 Simon Iosti and Denis Kuperberg. Eventually safe languages. In *Proceedings of DLT*, pages 192–205, 2019.
- 14 Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. Are good-for-games automata good for probabilistic model checking? In *Proceedings of LATA*, pages 453–465, 2014.
- 15 Denis Kuperberg and Anirban Majumdar. Computing the width of non-deterministic automata. *Logical Methods in Computer Science*, 15(4), 2019.
- 16 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Proceedings of ICALP*, pages 299–310, 2015.
- 17 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In *Proceedings of FSTTCS*, pages 66–77, 2011.
- 18 Karoliina Lehtinen and Martin Zimmermann. Good-for-games  $\omega$ -pushdown automata. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 689–702, 2020.
- 19 Christof Löding and Stefan Repke. Decidability Results on the Existence of Lookahead Delegates for NFA. In *Proceedings of FSTTCS*, pages 327–338, 2013.
- 20 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on  $\omega$ -words. *Theoretical Computer Science*, 32:321–330, 1984.
- 21 Domenic Quirl. Bachelor Thesis, supervised by Christof Löding, RWTH Aachen, 2018.



# A Framework for Consistency Algorithms

Peter Chini

TU Braunschweig, Germany  
p.chini@tu-braunschweig.de

Prakash Saivasan

The Institute of Mathematical Sciences, HBNI, Chennai, India  
prakashs@imsc.res.in

---

## Abstract

We present a framework that provides deterministic consistency algorithms for given memory models. Such an algorithm checks whether the executions of a shared-memory concurrent program are consistent under the axioms defined by a model. For memory models like SC and TSO, checking consistency is NP-complete. Our framework shows, that despite the hardness, fast deterministic consistency algorithms can be obtained by employing tools from fine-grained complexity.

The framework is based on a universal consistency problem which can be instantiated by different memory models. We construct an algorithm for the problem running in time  $\mathcal{O}^*(2^k)$ , where  $k$  is the number of write accesses in the execution that is checked for consistency. Each instance of the framework then admits an  $\mathcal{O}^*(2^k)$ -time consistency algorithm. By applying the framework, we obtain corresponding consistency algorithms for SC, TSO, PSO, and RMO. Moreover, we show that the obtained algorithms for SC, TSO, and PSO are optimal in the fine-grained sense: there is no consistency algorithm for these running in time  $2^{o(k)}$  unless the exponential time hypothesis fails.

**2012 ACM Subject Classification** Theory of computation → Concurrency; Theory of computation → Problems, reductions and completeness

**Keywords and phrases** Consistency, Weak Memory, Fine-Grained Complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.42

**Related Version** A full version is available via [20] or <https://arxiv.org/abs/2007.11398>.

## 1 Introduction

The paper at hand develops a framework for consistency algorithms. Given an execution of a concurrent program over a shared-memory system, consistency algorithms check whether the execution is consistent under the intended behavior of the memory. Our framework takes an abstraction of this intended behavior, a *memory model*, and yields a deterministic consistency algorithm for it. By applying the framework, we obtain provably optimal consistency algorithms for the well-known memory models SC [38], TSO, and PSO [1].

Checking consistency is central in the verification of shared-memory implementations. Such implementations promise programmers consistency guarantees according to a certain memory model. However, due to the complex and performance-oriented design, implementing shared memories is sensitive to errors and implementations may not provide the promised guarantees. Consistency algorithms test this. They take an execution over a shared-memory implementation, multiple sequences of read and write events, one for each thread. Then they check whether the execution is viable under the memory model, namely whether read and write events can be arranged in an interleaving that satisfies the axioms of the model.

In 1997, Gibbons and Korach [32] were the first ones that studied consistency checking as it is considered in this work. They focused on the basic memory model *Sequential Consistency* (SC) by Lamport [38]. In SC, read and write accesses to the memory are atomic making each write of a thread immediately visible to all other threads. Gibbons and Korach showed that



© Peter Chini and Prakash Saivasan;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 42; pp. 42:1–42:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

checking consistency in this setting is, in general, NP-complete. Moreover, they considered restrictions of the problem showing that even under the assumption that certain parameters like the number of threads are constant, the problem still remains NP-complete.

The SPARC memory models *Total Store Order* (TSO), *Partial Store Order* (PSO), and *Relaxed Memory Order* (RMO) were investigated by Cantin et al. in [15]. The authors showed that, like for SC, checking consistency for these models is NP-hard. Furbach et al. [31] extended the NP-hardness to almost all models appearing in the Steinke-Nutt hierarchy [46], a hierarchy developed for the classification of memory models. This yields NP-hardness results for memory models like *Causal Consistency* (CC) [37], *Pipelined RAM* (PRAM) [44], *Cache Consistency* [33] or variants of *Processor Consistency* [33, 4]. Bouajjani et al. [11] independently found that checking (variants of) CC for a given execution is NP-hard as well.

We approach consistency checking under the assumption of data-independence [11, 50, 10]. In fact, the behavior of a shared-memory implementation or a database does not depend on precise values in many practical applications [49, 3]. We can therefore assume that in a given execution, a value is written at most once. However, the NP-hardness of checking consistency under SC, TSO, and PSO carries over to the data-independent case [32, 31]. Deterministic consistency algorithms for these models will therefore face exponential running times. By employing a *fine-grained* complexity analysis, we show that one can still obtain consistency algorithms that have only a *mild* exponential dependence on certain parameters. Moreover, we show that the obtained algorithms are provably *optimal*.

Fine-grained complexity analyses are a task of *Parameterized Complexity* [30, 22, 24]. The goal of this new field within complexity theory is to measure the influence of certain parameters on a problem's complexity. In particular, if a problem is NP-hard, one can determine which parameter  $k$  of the problem still offers the opportunity for a fast deterministic algorithm. Such an algorithm runs in time  $f(k) \cdot \text{poly}(n)$ , where  $f$  is a computable function that only depends on the parameter, and  $\text{poly}(n)$  is a polynomial in the size of the input  $n$ . Problems admitting such algorithms lie in the class FPT of *fixed-parameter tractable* problems. The time-complexity of a problem in FPT is denoted by  $\mathcal{O}^*(f(k))$  since  $f(k)$  dominates. A fine-grained complexity analysis determines the precise function  $f$  that is needed to solve the problem. While finding upper bounds amounts to finding algorithms, lower bounds on  $f$  can be obtained from the *exponential time hypothesis* (ETH) [35]. It assumes that  $n$ -variable 3-SAT cannot be solved in time  $2^{o(n)}$ . Among other hardness assumptions, ETH is considered standard in parameterized complexity and was used to derive lower bounds for a variety of problems [22, 39, 21, 16]. A function  $f$  is *optimal* when upper and lower bound match.

Our contribution is a framework which yields consistency algorithms that are optimal in the fine-grained sense. Obtained algorithms run in time  $\mathcal{O}^*(2^k)$ , where  $k$  is the number of write events in the given execution. We demonstrate the applicability by obtaining corresponding consistency algorithms for SC, TSO, PSO, and RMO. Relying on the ETH, we prove that for the former three models, consistency cannot be checked in time  $2^{o(k)}$ . This shows that our framework yields optimal algorithms for these models. Moreover, we are significantly improving upon already existing deterministic algorithms that are usually based on a simple iteration running in time  $\mathcal{O}^*(k^k)$ . Note that considering other parameters like the number of threads, the number of events per thread, or the size of the underlying data domain yields W[1]-hard problems [42, 32] that are unlikely to admit FPT-algorithms [22, 24].

Our framework is based on a universal consistency problem that can be instantiated by a memory model of choice. We develop an algorithm for this universal problem running in time  $\mathcal{O}^*(2^k)$ . Then, any instance by a memory model automatically admits an  $\mathcal{O}^*(2^k)$ -time consistency algorithm. For the formulation of the problem, we rely on the formal framework

of Alglave [5] and Alglave et al. [6] for describing memory models in terms of relations. In fact, checking consistency then amounts to finding a particular *store order* [50] on the write events that satisfies various acyclicity constraints.

For solving the universal consistency problem, we show that instead of a store order we can also find a total order on the write events satisfying similar acyclicity constraints. The latter are algorithmically simpler to find. We develop a notion of *snapshot orders* that mimic total orders on subsets of write events. This allows for shifting from the relation-based domain of the problem to the subset lattice of writes. On this lattice, we can perform a dynamic programming which builds up total orders step by step and avoids an explicit iteration over such which would result in an  $\mathcal{O}^*(k^k)$ -time algorithm. Keeping track of the acyclicity constraints is achieved by so-called *coherence graphs*. The dynamic programming runs in time  $\mathcal{O}^*(2^k)$  which constitutes the time-complexity.

To apply the framework, we follow the formal description of SC, TSO, PSO, and RMO, given in [5, 6] and instantiate the universal consistency problem. Optimality of the algorithms for SC, TSO, and PSO is obtained from the ETH. To this end, we construct a reduction from 3-SAT to the corresponding consistency problem that generates only linearly many write events. The reduction transports the assumed lower bound on 3-SAT to consistency checking.

**Related Work.** In its general form, consistency checking is NP-hard for most memory models. Furbach et al. [31] show that LOCAL [2] is an exception. Checking consistency under LOCAL takes polynomial time. This also holds for *Cache Consistency* and PRAM if certain parameters of the consistency problem are assumed to be constant. In the case of data-independence, Bouajjani et al. [11] show that checking consistency under CC and variants of CC also takes polynomial time. Wei et al. [48] present a similar result for PRAM. In [50], Bouajjani et al. present practically efficient algorithms for the consistency problems of SC and TSO under data-independence. They rely on the polynomial-time algorithm for CC [11] and obtain a partial store order, which is completed by an enumeration. In theory, the enumeration has a worst-case time complexity of  $\mathcal{O}^*(k^k)$ . We avoid such an enumeration by a dynamic programming running in time  $\mathcal{O}^*(2^k)$ . Consistency checking for weaker and stronger notions of consistency, like *linearizability* [34], is considered in [26, 27, 25].

Instead of checking consistency for a single execution of a shared-memory implementation, there were efforts in verifying that all executions are consistent under a certain memory model. Alur et al. show in [7] that for SC, the problem is undecidable. This also holds for CC [11]. Under data-independence, the problem becomes decidable for CC [11]. Verifying *Eventual Consistency* [47] was shown to be decidable by Bouajjani et al. in [12]. There has also been work on other verification problems like reachability and robustness. Atig et al. show in [8] that, under TSO and PSO, reachability is decidable. In [9] the authors extend their results and present a relaxation of TSO with decidable reachability problem. Robustness against TSO was considered in [13] and shown to be PSPACE-complete. This also holds for POWER [40, 45], as shown in [23], and for partitioned global address spaces [14].

Parameterized complexity has been applied to other verification problems as well. Biswas and Enea [10] study the complexity of transactional consistency and obtain an FPT-algorithm in the size and the width of a history. This also yields an algorithm for the serializability problem, proven to be NP-hard by Papadimitriou [43] in 1979. A fine-grained algorithm for serializability under TSO was given in [28]. The authors of [29] present an FPT-algorithm for predicting atomicity violations as well as an intractability result. The parameterized complexity of data race prediction was considered in [42]. Fine-grained complexity analyses were conducted for reachability under *bounded context switching* on finite-state systems [17], and for reachability and liveness on parameterized systems [18, 19].

## 2 Preliminaries

To state our framework, we introduce some basic notions around memory models and the consistency problem. We mainly follow [6, 5, 50, 11]. Further, we give a short introduction into fine-grained complexity. For standard textbooks in this field, we refer to [30, 24, 22].

**Relations, Histories, and Memory Models.** We consider the consistency problem: given an execution of a concurrent program and a model of the shared memory, decide whether the execution adheres to the model. Formally, executions consist of *events* modeling write and read accesses to the shared memory. To define these, let  $Var$  be the finite set of variables of the program. Moreover, let  $Val$  be its finite data domain and  $Lab$  a finite set of labels. A *write event* is defined by  $w: wr(x, v)$ , where  $w \in Lab$  is a label,  $x \in Var$  is a variable, and  $v \in Val$  is a value. The set of write events is defined by  $WR = \{w: wr(x, v) \mid w \in Lab, x \in Var, v \in Val\}$ . A *read event* is given by  $r: rd(x, v)$ . The set of read events is denoted by  $RD$ . We define the set of all *events* by  $E = WR \cup RD$ . If it is clear from the context, we omit the label of an event. Given an event  $o \in E$ , we access the variable of  $o$  by  $var(o) \in Var$ . For a subset  $O \subseteq E$ , we denote by  $WR(O)$  and  $RD(O)$  the set of write and read events in  $O$ .

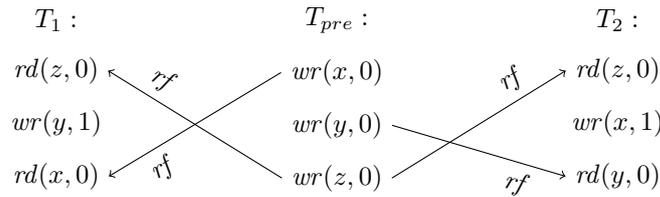
For modeling dependencies between events we use strict orders. Let  $O \subseteq E$  be a set of events. A *strict partial order* on  $O$  is an irreflexive, transitive relation over  $O^1$ . A *strict total order* is a strict partial order that is total. We often refer to the notions without mentioning that they are strict. Given two relations  $rel, rel' \subseteq O \times O$ , we denote by  $rel \circ rel'$  their composition, by  $rel^+$  the transitive closure, and by  $rel^{-1}$  the inverse. For variable  $x$ , we denote by  $rel_x$  the restriction of  $rel$  to events on  $x$ :  $rel_x = \{(o, o') \in rel \mid var(o) = var(o') = x\}$ .

Executions are modeled by *histories*. A *history* is a tuple  $h = \langle O, po, rf \rangle$ , where  $O \subseteq E$  is a set of events executed by the threads of the program. The *program order*  $po$  is a partial order on  $O$  which orders the events of a thread according to the execution. Typically, it is a union of total orders, one for each thread. The relation  $rf \subseteq WR(O) \times RD(O)$  is called *reads-from* relation. It specifies the write event providing the value for a read event in the history. Moreover, for each read event  $r \in RD(O)$  we have a write event  $w \in WR(O)$  such that  $(w, r) \in rf$  and if  $(w, r) \in rf$ , both events access the same variable.

► **Example 1.** Consider the history given in Figure 1. It consists of three threads  $T_1$ ,  $T_2$ , and  $T_{pre}$  that communicate via the variables  $x, y, z$  over the data domain  $\{0, 1\}$ . The set of events  $O$  is given by the events listed in the figure. Each thread processes from top to bottom indicating the program order  $po$ . Hence,  $po$  is the union of three total orders, one for each thread. For simplicity, we do not draw it. The reads-from relation is determined by the arrows labeled  $rf$ . The relation shows that each read event is linked to its corresponding write event. For instance, the two read events  $rd(z, 0)$  are linked to the write  $wr(z, 0)$ . Intuitively this means that in an actual execution, the threads  $T_1$  and  $T_2$  cannot start until  $T_{pre}$  finishes and writes  $wr(z, 0)$  to the memory since the correct value for  $z$  is not available earlier.

Note that in a history, we assume the reads-from relation  $rf$  to be given. This is due to the data-independence of shared-memory and database implementations in practice [49, 10, 3, 11, 50]. This means that the behavior of the implementation does not depend on actual values and in an execution, we may assume each value to be written at most once. From such an execution, we can simply read off the relation  $rf$ .

<sup>1</sup> Note that the relation has to be irreflexive. This separates it from a usual partial order.



■ **Figure 1** Example of a history. The program order is given implicitly by the arrangement of the events. For each thread,  $T_1, T_2$ , and  $T_{pre}$ , the program order progresses top to bottom. Formally, it is a union of the resulting three total orders. Arrows labeled by  $rf$  show the reads-from relation.

Our framework is compatible with histories that feature *initial writes*. These histories have a write event for each variable writing the initial value of that variable. Formally, these write events are smaller than all other events under program order. If a history  $h = \langle O, po, rf \rangle$  is fixed, we abuse notation and also use  $WR$  and  $RD$  to denote  $WR(O)$  and  $RD(O)$ . For a variable  $x$ , we write  $WR(x) = \{w \in WR \mid var(w) = x\}$  for the set of write events on  $x$  in  $h$ . Furthermore, we will later make use of the relation *po-loc*, defined by restricting  $po$  to events on the same variable:  $po-loc = \{(o, o') \in po \mid var(o) = var(o')\}$ .

A *memory model* is an abstraction of the memory behavior defining axioms that the relations in a history must adhere to. Formally, a *memory model*  $MM$  is a tuple  $MM = (po-mm, rf-mm)$ . The relation *po-mm*, also called *preserved program order*, is a subrelation of  $po$  describing the structure maintained by the memory model. The latter relation  $rf-mm$  is a subrelation of  $rf$ . It shows which write events are visible globally under  $MM$ .

**Fine-Grained Complexity.** For many memory models, the consistency problem is NP-hard [31, 32, 15, 11]. Hence, deterministic consistency algorithms usually face exponential running times. But exponents might only depend on certain parameters of the problem which still allow the algorithm for being fast. Finding such parameters is a task of *parameterized complexity*.

The basis of parameterized complexity are *parameterized problems*. That is, subsets  $P$  of  $\Sigma^* \times \mathbb{N}$ , where  $\Sigma$  is a finite alphabet. An input to  $P$  is of the form  $(x, k)$ , with  $k$  being called the *parameter*. A particularly interesting class of parameterized problems are the *fixed-parameter tractable* (FPT) problems. A problem  $P$  is FPT if it can be solved by a deterministic algorithm running in time  $f(k) \cdot |x|^{\mathcal{O}(1)}$ , where  $f$  is a computable function only dependent on  $k$ . The running time of such an algorithm is usually denoted by  $\mathcal{O}^*(f(k))$  to suppress the polynomial part. The class FPT is contained in the class  $W[1]$ . Problems that are  $W[1]$ -hard are considered intractable since they are unlikely to be FPT.

Given a fixed-parameter tractable problem  $P$ , finding an upper bound for  $f$  is achieved by constructing an algorithm for  $P$ . Lower bounds on  $f$  are usually obtained from the *exponential time hypothesis* (ETH) [35]. This standard hardness assumptions asserts that 3-SAT cannot be solved by an algorithm running in time  $2^{o(n)}$ , where  $n$  is the number of variables. A lower bound on  $f$  is then obtained by a suitable reduction from 3-SAT to  $P$ . We are interested in finding the *optimal*  $f$  for the consistency problem where upper and lower bound match. The search for such an  $f$  is referred to as *fine-grained complexity*.

### 3 Framework

We present our framework. Given a model describing the memory, the framework provides an (optimal) deterministic algorithm for the corresponding consistency problem. That is, whether a given history can be scheduled under the axioms imposed by the model. The obtained algorithm can then be used within a testing routine for concurrent programs.

At the heart of the framework is a universal consistency problem that can be instantiated with different memory models. We solve the problem by switching from a relation-based domain, where the problem is defined, to a subset-based domain. On the latter, we can then apply a dynamic programming which constitutes the desired deterministic algorithm.

### 3.1 Universal Consistency

The basis of our framework is a universal consistency problem which can be instantiated to simulate a particular memory model. For its formulation, we make use of a consistency notion that allows for the construction of a fast algorithm but deviates from the literature [5, 6, 50] at first sight. Therefore, it is proven in Section 4 that instantiating the problem with a particular memory model yields the correct notion of consistency.

We clarify our notion of consistency. Intuitively, a history is consistent under a memory model if it can be scheduled such that certain axioms defined by the model are satisfied. Following the formal framework of [5, 6], finding such a schedule amounts to finding a particular order of the write events that satisfies acyclicity requirements imposed by the axioms. Formally, let  $h = \langle O, po, rf \rangle$  be a history and let MM be a memory model described by the tuple  $(po\text{-}mm, rf\text{-}mm)$ . Then  $h$  is called MM-consistent if there exists a strict total order  $tw$  on the write events  $WR$  of  $h$  such that the graphs

$$G_{loc} = (O, po\text{-}loc \cup rf \cup tw \cup cf) \quad \text{and} \quad G_{mm} = (O, po\text{-}mm \cup rf\text{-}mm \cup tw \cup cf)$$

are both acyclic. Here, the *conflict relation*  $cf$  is defined by  $cf = rf^{-1} \circ \bigcup_{x \in Var} tw_x$ . Phrased differently,  $(r, w) \in cf$  if  $r$  is a read event on a variable  $x$ ,  $w$  is a write event on  $x$ , and there is a write event  $w'$  on  $x$  such that  $(w', r) \in rf$  and  $(w', w) \in tw$ .

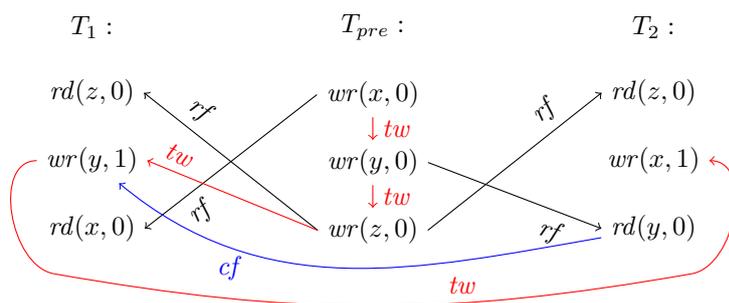
The acyclicity of  $G_{loc}$  is called *uniprocessor* requirement [5] or *memory coherence* for each location [15]. Roughly, it demands that an order among writes to the same location that can be extracted from the history, is kept in  $tw$ . The second acyclicity requirement in the definition resembles the underlying memory model MM. If  $G_{mm}$  is acyclic, the history can be scheduled adhering to the axioms defined by MM.

► **Example 2.** Consider the history given in Example 1. We check consistency under the simple memory model SC. As we will see later in Section 4.2, SC is defined by the tuple  $(po\text{-}sc, rf\text{-}sc) = (po, rf)$ . For checking consistency, we need to construct the graphs  $G_{loc}$  and  $G_{sc}$ . To this end, we fix a total order  $tw$  on the write events. It is shown as the red edges labeled by  $tw$  in Figure 2. Formally, the strict total order  $tw$  is the transitive closure of these edges. The next step is to determine the conflict relation  $cf$ . It contains two edges. There is an edge  $rd(y, 0) \rightarrow wr(y, 1)$ , shown in blue in Figure 2. This is due to the inverted  $rf$ -edge  $rd(y, 0) \rightarrow wr(y, 0)$  and the  $tw$ -edge  $wr(y, 0) \rightarrow wr(y, 1)$ . Note that the latter edge exists in  $tw$  (transitive closure) and connects writes to the same variable  $y$  which is mandatory for  $cf$ . The second  $cf$ -edge  $rd(x, 0) \rightarrow wr(x, 1)$  is obtained similarly but is not shown in the figure.

According to the chosen memory model SC, the graph in Figure 2 shows a subgraph of  $G_{sc} = (O, po \cup rf \cup tw \cup cf)$ . In fact, only the second conflict edge is missing. But we already obtain a cycle in this graph which traverses as follows:

$$rd(y, 0) \xrightarrow{cf} wr(y, 1) \xrightarrow{tw} wr(x, 1) \xrightarrow{po} rd(y, 0).$$

This constitutes a cycle in  $G_{sc}$  and shows that the chosen total order  $tw$  does not lead to acyclic graphs and is therefore not a witness for consistency. However, any total order on the write events will cause a cycle implying that the history is not SC-consistent.



■ **Figure 2** A subgraph of  $G_{sc}$ . The total order  $tw$  is the transitive closure of the red edges. The blue edge is part of the conflict relation  $cf$ . One  $cf$ -edge, namely  $rd(x,0) \rightarrow wr(x,1)$ , is missing. The graph contains a cycle showing that the underlying history is not SC-consistent.

Our definition of consistency deviates from the literature in two aspects. First, we demand a total order  $tw$  instead of a *store order*, a partial order that is total on writes to the same location [5, 6, 50]. In Section 4 we will show that the resulting notions of consistency are equivalent. A further difference is that we do not explicitly test for *out of thin air values* [41]. For the majority of memory models considered in this work, the test is not necessary as it is implied by the acyclicity of  $G_{loc}$  and  $G_{mm}$ . But it can easily be added when needed.

We are ready to state the universal consistency problem. To this end, let MM be a fixed memory model. Given a history  $h$ , the problem asks whether  $h$  is MM-consistent.

*MM-Consistency*

**Input:** A history  $h = \langle O, po, rf \rangle$ .

**Question:** Is  $h$  MM-consistent?

Instantiations of the problem by well-known memory models like SC or TSO are typically NP-hard [32, 31]. However, we are interested in a deterministic algorithm for *MM-Consistency*. While we cannot avoid an exponential running time for such an algorithm, a fine-grained complexity analysis can determine the *optimal* exponential dependence. Many parameters of *MM-Consistency* like the number of threads, the maximum size per thread, or the size of the data domain yield parameterizations that are W[1]-hard [42, 32]. Therefore, we conduct a fine-grained analysis for the parameter  $k = |WR|$ , the number of writes in  $h$ . The main finding is an algorithm for *MM-Consistency* running in time  $\mathcal{O}^*(2^k)$ . The optimality of this approach is shown in Section 5 by a complementing lower bound. We formally state the upper bound in the following theorem. There,  $n = |O|$  denotes the number of events in  $h$ .

► **Theorem 3.** *The problem MM-Consistency can be solved in time  $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$ .*

Note that an algorithm for *MM-Consistency* running in time  $\mathcal{O}^*(k^k)$  is immediate. One can iterate over all total orders of  $WR$  and check the acyclicity of  $G_{loc}$  and  $G_{mm}$  in polynomial time. Since we cannot afford this iteration in  $\mathcal{O}^*(2^k)$ , improving the running time needs an alternative approach and further technical development that we summarize in Section 3.2.

### 3.2 Algorithm

We present the upper bound for *MM-Consistency* as stated in Theorem 3. Our algorithm is a dynamic programming. It switches from the domain of total orders to subsets of write events and iterates over the latter. The crux is that for a particular subset we do not need to

remember a precise order. In fact, we only need to store that it can be ordered by a so-called *snapshot order* that mimics total orders on subsets. Not having a precise order at hand yields a disadvantage: we cannot just test both acyclicity requirements in the end. Instead, we perform an acyclicity test on a *coherence graph* in each step of the iteration. These graphs carry enough information to ensure acyclicity as it is required by *MM-Consistency*.

We begin our technical development by introducing *snapshot orders*. Intuitively, these simulate total orders of the write events on subsets of writes. Given a subset, a *snapshot order* consists of two parts: a total order on the subset and a partial order. The latter expresses that the complement of the given set precedes the subset but is yet unordered.

► **Definition 4.** Let  $V \subseteq WR$ . A *snapshot order* on  $V$  is a union  $tw[V] = t[V] \cup r[V]$ .

The relation  $t[V]$  is a strict total order on  $V$  and  $r[V] = \{(\bar{v}, v) \mid \bar{v} \in \bar{V}, v \in V\}$  arranges that the elements of  $\bar{V}$  are smaller than the elements of  $V$ . By  $\bar{V}$ , we denote the complement of  $V$  in the write events,  $\bar{V} = WR \setminus V$ . Note that  $r[V]$  does not impose an order among  $\bar{V}$ .

A snapshot order is indeed a strict partial order. Even more, when the considered set is the whole write events  $WR$ , a snapshot order  $tw[WR]$  is a total order on  $WR$ . Therefore, *MM-consistency* can be checked by finding a snapshot order on  $WR$  satisfying both acyclicity requirements. The advantage of this formulation is that we can construct such an order from snapshot orders on subsets. Technically, we parameterize<sup>2</sup> the problem along all  $V \subseteq WR$ .

For the acyclicity requirements, we need a similar parameterization. To this end, let  $V \subseteq WR$  be a subset and  $tw[V]$  a snapshot order on  $V$ . We parameterize the above graphs  $G_{loc}$  and  $G_{mm}$  via exchanging the total order by the snapshot order:

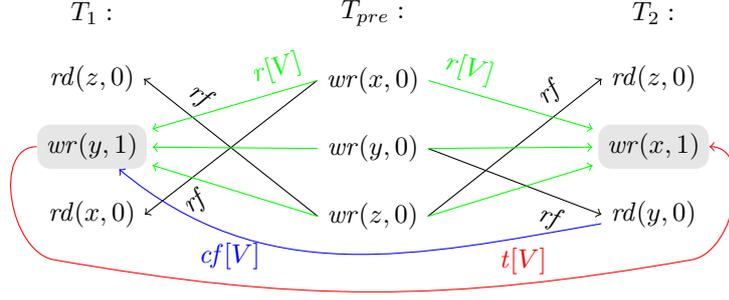
$$\begin{aligned} G_{loc}(tw[V]) &= (O, po\text{-}loc \cup rf \cup tw[V] \cup cf[V]), \\ G_{mm}(tw[V]) &= (O, po\text{-}mm \cup rf\text{-}mm \cup tw[V] \cup cf[V]). \end{aligned}$$

As above, the conflict relation is defined by  $cf[V] = rf^{-1} \circ \bigcup_{x \in Var} tw[V]_x$ . Note that for a snapshot order  $tw[WR]$  on the whole set of write events, the resulting graphs  $G_{loc}(tw[WR])$  and  $G_{mm}(tw[WR])$  are exactly those appearing in the acyclicity requirement.

► **Example 5.** We reconsider the history of Examples 1 and 2. Our goal is to construct the graph  $G_{sc}(tw[V])$  along a snapshot order  $tw[V]$ . To this end, we first fix a set  $V$ . Let  $V = \{wr(y, 1), wr(x, 1)\}$ . The set is shown in Figure 3 by the gray highlighted write events. As a snapshot order we chose  $tw[V] = t[V] \cup r[V]$ , where  $t[V]$  consists of only one edge:  $wr(y, 1) \rightarrow wr(x, 1)$ . Note that this is a total order on  $V$ . The edge is shown in Figure 3, it is marked red and labeled by  $t[V]$ . The relation  $r[V]$  is fixed by definition. It contains an edge from each write event in  $\bar{V}$  to each write event in  $V$ . These are marked green in Figure 3. To construct  $G_{sc}(tw[V])$  it is left to determine the relation  $cf[V]$ . The relation contains two edges,  $rd(y, 0) \rightarrow wr(y, 1)$  and  $rd(x, 0) \rightarrow wr(x, 1)$ . We show the former edge in Figure 3 as well. The latter is omitted to ease readability.

Note that the graph clearly shows that the set  $V$  is totally ordered by  $t[V]$  while the set  $\bar{V}$  is not. The only information that we obtain, from  $r[V]$ , is that the write events in  $\bar{V}$  are *smaller* than the elements in  $V$ . In this case, this is already enough to obtain a cycle. This means that each total order on write events that contains  $tw[V]$  cannot witness *SC-consistency*. Note that the total order of Example 2 is such an order.

<sup>2</sup> The parameterization here does not refer to parameterized complexity.



■ **Figure 3** The graph  $G_{sc}(tw[V])$  with set  $V = \{wr(y, 1), wr(x, 1)\}$ , highlighted gray. The snapshot order  $tw[V]$  is given as the union of the total order  $t[V]$ , marked red, and the partial order  $r[V]$ , marked green. The relation  $cf[V]$  consists of two edges,  $rd(y, 0) \rightarrow wr(y, 1)$ , shown in blue, and  $rd(x, 0) \rightarrow wr(x, 1)$ , not shown in the figure.

Now we have the tools to state the parameterization of *MM-Consistency* along subsets of write events. This allows for leaving the domain of total orders and switch to subsets instead. To this end, we define a table  $T$  with a Boolean entry  $T[V]$  for each  $V \subseteq WR$ . Entry  $T[V]$  will be 1, if there is a snapshot order on  $V$  satisfying the acyclicity requirement on both parameterized graphs. Otherwise,  $T[V]$  will evaluate to 0. Formally,  $T[V]$  is defined by

$$T[V] = \begin{cases} 1, & \text{if } \exists \text{ snapshot ord. } tw[V] : G_{loc}(tw[V]) \text{ and } G_{mm}(tw[V]) \text{ are acyclic,} \\ 0, & \text{otherwise.} \end{cases}$$

The following lemma relates *MM-Consistency* to the table  $T$ . It is crucial in our development as it states the correctness of the constructed parameterization. The proof follows from the beforehand definitions and the fact that a snapshot order on  $WR$  is already total.

► **Lemma 6.** *History  $h$  is MM-consistent if and only if  $T[WR] = 1$ .*

We are now left with the problem of evaluating the entry  $T[WR]$ . Our approach is to set up a recursion among the entries of  $T$  and evaluate it via a bottom-up dynamic programming. The recursion will explain how entries of subsets are aggregated to compute entries of larger sets. In fact, write events are added element by element: the recursion shows how an entry  $T[V]$  can be utilized to compute the entry of an enlarged set  $V \cup \{v\}$ , where  $v \in \bar{V}$ .

When passing from  $T[V]$  to  $T[V \cup \{v\}]$ , we need to provide a snapshot order on  $V \cup \{v\}$  that satisfies the acyclicity requirements. A snapshot order on  $V$  can always be extended to a snapshot order on  $V \cup \{v\}$ : we insert  $v$  as new minimal element in the contained total order. But we need to keep track of whether the acyclicity is compatible with the new minimal element  $v$ . To this end, we perform acyclicity tests on *coherence graphs*. These do not depend on a snapshot order and solely rely on the fact that  $v$  is the new minimal element. This will later allow for an evaluation of the table without touching precise orders.

► **Definition 7.** Let  $V \subseteq WR$  and  $v \in \bar{V}$ . The *coherence graphs* of  $V$  and  $v$  are defined by

$$\begin{aligned} G_{loc}[V, v] &= (O, po\text{-}loc \cup rf \cup r[V, v] \cup cf[V, v]), \\ G_{mm}[V, v] &= (O, po\text{-}mm \cup rf\text{-}mm \cup r[V, v] \cup cf[V, v]). \end{aligned}$$

In the definition, relation  $r[V, v]$  expresses that  $\overline{V \cup \{v\}}$  is smaller than  $V \cup \{v\}$  and that  $v$  is the minimal element in  $V \cup \{v\}$ . Formally, it is given by  $r[V, v] = r[V \cup \{v\}] \cup \{(v, w) \mid w \in V\}$ . The conflict relation is defined by  $cf[V, v] = rf^{-1} \circ \bigcup_{x \in Var} r[V, v]_x$ .

Coherence graphs are key for the recursion among the entries of  $T$ . Assume we are given a snapshot order  $tw[V]$  on  $V$  meeting the acyclicity requirements of  $T$  and we extend it to a snapshot order  $tw[V']$  on  $V' = V \cup \{v\}$ , as above - by inserting  $v$  as minimal element of  $V'$ . We show that each potential cycle in  $G_{loc}(tw[V'])$  or  $G_{mm}(tw[V'])$  either implies a cycle in a coherence graph  $G_{loc}[V, v]$  or  $G_{mm}[V, v]$  or in one of the graphs  $G_{loc}(tw[V])$  or  $G_{mm}(tw[V])$ . If  $T[V] = 1$ , we can assume the latter graphs to be acyclic. Moreover, if we have checked that the coherence graphs are acyclic as well, we obtain that  $T[V'] = 1$ . Hence, a recursion should check whether  $T[V] = 1$  and whether the corresponding coherence graphs are acyclic.

We formulate the recursion in the subsequent lemma. Note that it is a top-down formulation that only refers to non-empty subsets of write events. An evaluation of the base case is immediate. Entry  $T[\emptyset]$  is evaluated to 1 if  $G_{loc}(\emptyset) = (O, po\text{-}loc \cup rf)$  and  $G_{mm}(\emptyset) = (O, po\text{-}mm \cup rf\text{-}mm)$  are both acyclic. Otherwise it is evaluated to 0.

► **Lemma 8.** *Let  $V \subseteq WR$  be a non-empty subset. Entry  $T[V]$  admits the following recursion:*

$$T[V] = \bigvee_{v \in V} (G_{loc}[V \setminus \{v\}, v] \text{ acyclic}) \wedge (G_{mm}[V \setminus \{v\}, v] \text{ acyclic}) \wedge T[V \setminus \{v\}].$$

We interpret the expression  $(G_{loc}[V \setminus \{v\}, v] \text{ acyclic})$  as a predicate evaluating to 1 if the graph is acyclic and to 0 otherwise. Hence, the recursion requires the existence of a write event  $v \in V$  such that both coherence graphs are acyclic and entry  $T[V \setminus \{v\}]$  evaluates to 1. A proof of Lemma 8 is provided in the full version of the paper.

With the recursion at hand we can evaluate the table  $T$  by a dynamic programming. To this end, we store already computed entries and look them up when needed. An entry  $T[V]$  is evaluated as follows. We branch over all write events  $v \in V$  and test whether the coherence graphs  $G_{loc}[V \setminus \{v\}, v]$  and  $G_{mm}[V \setminus \{v\}, v]$  are acyclic. Then, we look up whether  $T[V \setminus \{v\}] = 1$ . If all three queries are positive, we store  $T[V] = 1$ . Otherwise,  $T[V] = 0$ .

The complexity estimation of Theorem 3 is obtained as follows. The table has  $2^k$  many entries that we evaluate, which constitutes the exponential factor. For each entry  $T[V]$ , we branch over at most  $k$  write events  $v \in V$ . Looking up the value of  $T[V \setminus \{v\}]$  can be done in constant time. The following lemma shows that  $\mathcal{O}(k \cdot n^2)$  time suffices to construct the coherence graphs and to check them for acyclicity. The latter checks are based on Kahn's algorithm [36] for finding a topological sorting. This completes the proof of Theorem 3.

► **Lemma 9.** *Let  $V \subseteq WR$  and  $v \in \bar{V}$ . Constructing the coherence graphs  $G_{loc}[V, v]$  and  $G_{mm}[V, v]$  and testing both for acyclicity can be done in time  $\mathcal{O}(k \cdot n^2)$ .*

## 4 Instantiating the Framework

We show the applicability of our framework and obtain consistency algorithms for the memory models SC, TSO, PSO, and RMO. To this end, we first need to show that our notion of consistency coincides with the notion of consistency used in the literature for these models. This ensures that the obtained algorithms really solve the correct problem. Once this is achieved, we can directly apply the framework to SC, TSO, and PSO. For RMO, we show how the framework can be slightly modified to also capture this more relaxed model.

### 4.1 Validity

Consistency, as it is considered in the literature, is also known as *validity* [5, 6]. We use the latter name to avoid confusion with our notion of consistency. Before we show that both notions actually coincide, we formally define validity. The definition is based on *store*

orders [5, 6, 50] (also known as *coherence orders*). Given a history  $h = \langle O, po, rf \rangle$ , a *store order*  $ww \subseteq WR \times WR$  takes the form  $ww = \bigcup_{x \in Var} ww_x$  so that each  $ww_x$  is a strict total order on  $WR(x)$ . Phrased differently, store orders are unions of total orders on writes to the same variable. Note that, in contrast to a total order on  $WR$ , a store order does not have any edge between write events referring to distinct variables.

Validity is similar to consistency. But instead of a total order, the acyclicity requirements need to be satisfied by a store order. Let MM be a memory model described by  $(po-mm, rf-mm)$ . A history  $h = \langle O, po, rf \rangle$  is *MM-valid* if there exists a store order so that

$$G_{loc}^{ww} = (O, po-loc \cup rf \cup ww \cup fr) \quad \text{and} \quad G_{mm}^{ww} = (O, po-mm \cup rf-mm \cup ww \cup fr)$$

are acyclic. The *from-read* relation is defined by  $fr = rf^{-1} \circ ww$ . Note that the definition, as in the case of consistency above, omits checking for out of thin air values. We will later add an explicit test for memory models that require it. This will not affect the complexity.

We show the equivalence of validity and consistency. To this end, we need to prove that a store order can be replaced by a total order on the write events while acyclicity is preserved. The following lemma states the result. It is crucial for the applicability of our framework.

► **Lemma 10.** *A history  $h$  is MM-valid if and only if it is MM-consistent.*

Before we give the proof of Lemma 10, we need an auxiliary statement. It shows that a store order  $ww$  in  $G_{loc}^{ww}$  can be replaced by any linearization of  $ww$  without affecting acyclicity. Phrased differently, any total order  $tw$  on the write events that contains  $ww$  can be inserted into the graph  $G_{loc}^{ww}$  - it will still be acyclic. We state the corresponding lemma.

► **Lemma 11.** *Let  $h = \langle O, po, rf \rangle$  be a history,  $ww$  a store order, and  $tw$  a total order on  $WR$  such that  $ww \subseteq tw$ . If  $G_{loc}^{ww}$  is acyclic, then so is  $G_{loc}^{tw} = (O, po-loc \cup rf \cup tw \cup fr)$ .*

The proof of Lemma 11 is given in the full version. We turn to the proof of Lemma 10.

**Proof of Lemma 10.** First assume that  $h = \langle O, po, rf \rangle$  is MM-valid. Then there is a store order  $ww$  such that  $G_{loc}^{ww}$  and  $G_{mm}^{ww}$  are acyclic. Consider the edges of the latter graph. They form a relation  $ord-mm = po-mm \cup rf-mm \cup ww \cup fr$ . Since  $G_{mm}^{ww}$  is acyclic, the transitive closure  $ord-mm^+$  is a strict partial order on  $O$ . Hence, there exists a linear extension, a strict total order  $L$  containing  $ord-mm^+$ . We define  $tw = L \cap WR \times WR$ . Then,  $tw$  is a total order on  $WR$  and we have  $ww \subseteq L \cap WR \times WR = tw$ . We show that  $G_{loc}$  and  $G_{mm}$  are acyclic. Note that the latter refer to the graphs from the definition of consistency.

The store order  $ww$  is contained in  $tw$ . Hence, we obtain that  $ww_x \subseteq tw_x$  for each variable  $x \in Var$ . This implies that  $ww_x = tw_x$  since  $ww_x$  is total on  $WR(x)$ . We can deduce  $ww = \bigcup_{x \in Var} ww_x = \bigcup_{x \in Var} tw_x$  and thus  $cf = rf^{-1} \circ \bigcup_{x \in Var} tw_x = rf^{-1} \circ ww = fr$ .

Since  $fr = cf$ , we get the acyclicity of  $G_{loc} = G_{loc}^{tw}$  from Lemma 11. The acyclicity of  $G_{mm}$  follows since its edges  $po-mm \cup rf-mm \cup tw \cup cf$  form a subrelation of  $L$ . A cycle would mean that  $L$  has a reflexive element, but  $L$  is a strict order. Hence,  $h$  is MM-consistent.

For the other direction, assume that  $h$  is MM-consistent. By definition, there is a total order  $tw$  on  $WR$  such that  $G_{loc}$  and  $G_{mm}$  are acyclic. We construct the store order  $ww = \bigcup_{x \in Var} tw_x$ . Note that, since  $tw_x$  is total on  $WR(x)$ ,  $ww$  is indeed a store order and we have  $ww \subseteq tw$ . We show that  $G_{loc}^{ww}$  and  $G_{mm}^{ww}$  are acyclic. In fact, we have that  $fr = rf^{-1} \circ ww = cf$ . This implies that  $G_{loc}^{ww}$  and  $G_{mm}^{ww}$  are subgraphs of  $G_{loc}$  and  $G_{mm}$ , respectively. Hence, the two graphs are acyclic and  $h$  is MM-valid. ◀

## 4.2 Instances

We apply the algorithmic framework to the mentioned memory models and obtain (optimal) deterministic algorithms for their corresponding validity/consistency problem. To this end, we employ the formal description of these models given in [5, 6].

**Sequential Consistency.** *Sequential Consistency* (SC) is a basic memory model, first defined by Lamport in [38]. Intuitively, SC strictly follows the given program order and flushes each issued write immediately to the memory so that it is visible to all other threads.

Formally, SC is described by the tuple  $SC = (po-sc, rf-sc)$  with  $po-sc = po$  and  $rf-sc = rf$ . Hence, it employs the full program order and reads-from relation, making the uniprocessor test on  $G_{loc}$  obsolete. However, our framework still applies. It yields an algorithm for the corresponding validity/consistency problem running in time  $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$ . We show in Section 5 that the obtained algorithm is optimal under ETH.

**Total Store Ordering.** The SPARC memory model *Total Store Order* (TSO) [1] resembles a more relaxed memory behavior. Instead of flushing writes immediately to the memory, like in SC, each thread has an own FIFO buffer and issued writes of that thread are pushed into the buffer. Writes in the buffer are only visible to the owning thread. If the owner reads a certain variable, it first looks through the buffer and reads the latest issued write on that variable. This is called *early read*. At some nondeterministic point, the buffer is flushed to the memory, making the writes visible to other threads as well.

The formal description of TSO is given by the tuple  $TSO = (po-tso, rf-tso)$ , where  $po-tso = po \setminus WR \times RD$  is a relaxation of the program order, containing no write-read pairs. The relation  $rf-tso = rf_e$  is a restriction of  $rf$  to write-read pairs from different threads:

$$rf_e = \{(w, r) \in rf \mid (w, r) \notin po, (r, w) \notin po\}.$$

Unlike in the case of SC, we do not have the full program order and reads-from relation at hand. Hence, the uniprocessor test is essential. Applying the framework yields an algorithm for the validity/consistency problem of TSO running in time  $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$ . The optimality of the obtained algorithm is shown in Section 5.

**Partial Store Ordering.** The second SPARC model that we consider is *Partial Store Order* (PSO) [1]. It is weaker than TSO since writes to different locations issued by a thread may not arrive at the memory in program order. Intuitively, in PSO each thread has a buffer per variable where the corresponding writes to the variable are pushed. Like for TSO, threads can read early from their buffers and the buffers are, at some point, flushed to the memory.

Formally, PSO is captured by the tuple  $PSO = (po-pso, rf-pso)$ . Here, the relation  $po-pso = po \setminus (WR \times RD \cup WR \times WR)$  takes away the write-read pairs and the write-write pairs from the program order and, like for TSO, we have  $rf-pso = rf_e$ . Hence, we can apply our framework and obtain an  $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$ -time algorithm. The obtained algorithm is optimal.

**Relaxed Memory Order.** We extend the framework to also capture SPARC's *Relaxed Memory Order* (RMO) [1]. The model needs an explicit out of thin air test and allows for so-called *load-load hazards*. We show how both modifications can be built into the framework without affecting the complexity of the resulting consistency algorithm.

The model RMO relies on an additional *dependency relation* resembling address and data dependencies among events in an execution of a program. For instance, if a read event has influence on the value written by a subsequent write event. We assume that the *dependency*

relation  $dp$  is given along with a history  $h = \langle O, po, rf \rangle$  and is a subrelation of  $po \cap (RD \times O)$ . The latter means that  $dp$  always starts in a read event. With the relation at hand we can perform an out of thin air test. In fact, such a test [5] requires that  $(O, dp \cup rf)$  is acyclic. This can be checked by Kahn's algorithm [36] in time  $\mathcal{O}(n^2)$ . Hence, the test can be added to the framework without increasing the time complexity of the obtained consistency algorithm.

Load-load hazards are allowed by RMO. These occur when two reads of the same variable are scheduled not following the program order. To obtain an algorithm from the framework in this case, we need to weaken the uniprocessor check [5]. In fact, we replace the relation  $po-loc$  by  $po-loc_{llh} = po-loc \setminus RD \times RD$  and require that the graph  $G_{loc-llh} = (O, po-loc_{llh} \cup rf \cup tw \cup cf)$  is acyclic. The correctness of the framework is ensured since Lemma 10 still holds in this setting. Moreover, the running time of the resulting algorithm is not affected.

With these modifications, we can obtain a consistency algorithm for RMO. Formally,  $RMO = (po-rmo, rf-rmo)$  where  $po-rmo = dp$  and  $rf-rmo = rf_e$ . Applying the framework with out of thin air test and  $G_{loc-llh}$  yields a consistency algorithm running in  $\mathcal{O}(2^k \cdot k^2 \cdot n^2)$ .

## 5 Lower Bounds

We show that the framework provides optimal consistency algorithms for SC, TSO, and PSO. To this end, we employ the ETH and prove that checking consistency under these three memory models cannot be achieved in subexponential time  $2^{o(k)}$ . Since the algorithms obtained in Section 4 match the lower bound, they are indeed optimal.

We begin with the lower bound for SC-Consistency. For its proof, we rely on a characterization of the ETH, known as the *Sparsification Lemma* [35]. It states that ETH is equivalent to the assumption that 3-SAT cannot be solved in time  $2^{o(n+m)}$ , where  $n$  is the number of variables and  $m$  is the number of clauses of the input formula. To transport the lower bound to consistency checking, we construct a polynomial-time reduction from 3-SAT to SC-Consistency which controls the number of writes  $k$ . Technically, for a given formula  $\varphi$ , the reduction yields a history  $h_\varphi$  that has only  $k = \mathcal{O}(n + m)$  many write events and is SC-consistent if and only if  $\varphi$  is satisfiable. By invoking the reduction, an  $2^{o(k)}$ -time algorithm for SC-Consistency, would yield an  $2^{o(n+m)}$ -time algorithm for 3-SAT, contradicting the ETH.

► **Theorem 12.** *SC-Consistency cannot be solved in time  $2^{o(k)}$  unless ETH fails.*

It is left to construct the reduction. Let  $\varphi$  be a 3-SAT-instance over the variables  $X = \{x_1, \dots, x_n\}$  and with clauses  $C_1, \dots, C_m$ . Moreover, let  $L$  denote the set of literals. We construct a history  $h_\varphi$  the number of writes of which depends linearly on  $n + m$ .

The main idea of the reduction is to mimic an evaluation of  $\varphi$  by an interleaving of the events in  $h_\varphi$ . To this end, we divide evaluating  $\varphi$  into three steps: (1) choose an evaluation of the variables, (2) evaluate the literals accordingly, and (3) check whether the clauses are satisfied. For each of these steps we have separate threads taking care of the task. Scheduling them in different orders will yield different evaluations. An overview is given in Figure 4.

Figure 4 presents  $h_\varphi$  as a collection of threads. The program order is obtained from reading threads top to bottom. The reads-from relation is given since each value is written at most once to a variable. Hence, there is always a unique write event providing the read value.

We elaborate on the details of the reduction. For realizing Step (1), we construct two threads,  $T_0(x)$  and  $T_1(x)$ , for each variable  $x \in X$ . These mimic an evaluation of the variable and consist of only one write event. Thread  $T_0(x)$  writes 0 to  $x$ , thread  $T_1(x)$  writes 1. If  $T_0(x)$  gets scheduled before  $T_1(x)$ , variable  $x$  is evaluated to 1 and to 0 otherwise. Hence, the thread that is scheduled later will determine the actual evaluation of  $x$ .

$T_0(x) :$	$T_1(x) :$	$T_0(\ell) :$	$T_1(\ell) :$	$T^1(C) :$	$T^2(C) :$	$T^3(C) :$
$wr(x, 0)$	$wr(x, 1)$	$rd(x, 0)$	$rd(x, 1)$	$rd(\ell_3, 0)$	$rd(\ell_1, 0)$	$rd(\ell_2, 0)$
		$wr(\ell, c)$	$wr(\ell, d)$	$rd(\ell_1, 1)$	$rd(\ell_2, 1)$	$rd(\ell_3, 1)$
		$rd(x, 0)$	$rd(x, 1)$			

■ **Figure 4** Parts of the history  $h_\varphi$  for a variable  $x \in X$ , a literal  $\ell \in L$ , and a clause  $C = \ell_1 \vee \ell_2 \vee \ell_3$ . Values of  $c$  and  $d$  depend on  $\ell$ . If  $\ell = x$ , then  $c = 0, d = 1$ . Otherwise,  $c = 1, d = 0$ .

In Step (2), we propagate the evaluation of the variables to the literals. To this end, we construct two threads for each literal  $\ell \in L$ . Let  $\ell = x/\neg x$  be a literal on variable  $x \in X$ . The first thread  $T_0(\ell)$  is responsible for evaluating  $\ell$  when  $x$  is evaluated to 0. It first performs a read event  $rd(x, 0)$ , followed by  $wr(\ell, c)$  and  $rd(x, 0)$ . The value  $c$  depends on the literal: if  $\ell = x$ , then  $c = 0$ . Otherwise  $c = 1$ . Note that the read events guard the write event. This ensures that  $T_0(\ell)$  can only run if  $x$  is already evaluated to 0 and once  $T_0(\ell)$  is running, the evaluation of  $x$  cannot change until the thread finishes. Thread  $T_1(\ell)$  behaves similar. It evaluates the literal  $\ell$  when  $x$  is evaluated to 1. Both threads cannot interfere. Like for the variables, the later scheduled thread determines the actual evaluation of the literal.

It is left to evaluate the clauses. For a clause  $C = \ell_1 \vee \ell_2 \vee \ell_3$ , we have threads  $T^1(C)$ ,  $T^2(C)$ , and  $T^3(C)$  as shown in Figure 4. It is the task of these threads to ensure that at least one literal in  $C$  evaluates to 1. To see this, assume we have the contrary, an evaluation of the variables (and the literals) such that  $\ell_1, \ell_2$ , and  $\ell_3$  evaluate to 0. Due to the construction,  $\ell_1$  storing 0 implies that  $wr(\ell_1, 1)$  preceded the write event  $wr(\ell_1, 0)$ . Hence, the read event  $rd(\ell_1, 1)$  in  $T^1(C)$  must have already been scheduled. In particular, it has to occur before  $rd(\ell_1, 0)$  in  $T^2(C)$ . Since  $\ell_2$  and  $\ell_3$  also store 0, we get a similar dependency among their reads:  $rd(\ell_2, 1)$  occurs before  $rd(\ell_2, 0)$  and  $rd(\ell_3, 1)$  occurs before  $rd(\ell_3, 0)$ . Due to program order, we obtain a dependency cycle involving all these reads:

$$rd(\ell_1, 1) \rightarrow rd(\ell_1, 0) \rightarrow rd(\ell_2, 1) \rightarrow rd(\ell_2, 0) \rightarrow rd(\ell_3, 1) \rightarrow rd(\ell_3, 0) \rightarrow rd(\ell_1, 1).$$

An arrow  $r \rightarrow r'$  means that  $r$  has to precede  $r'$  in an interleaving of the events in  $h_\varphi$ . Since cycles cannot occur in an interleaving, the threads can only be scheduled properly when a satisfying assignment is given. The construction of a proper schedule is subtle. We provide details in the full version of the paper. The following lemma states the correctness.

► **Lemma 13.** *Formula  $\varphi$  is satisfiable if and only if the history  $h_\varphi$  is SC-consistent.*

Clearly,  $h_\varphi$  can be constructed in polynomial time. We determine the number of write events. For each variable  $x \in X$  and each literal  $\ell \in L$ , we introduce two write events. Hence,  $k = 2 \cdot n + 2 \cdot |L|$ . Since there are at most  $3 \cdot m$  many literals in  $\varphi$ , we get that  $k$  is bounded by  $2 \cdot n + 6 \cdot m$ , a number linear in  $n + m$ . This finishes the proof of Theorem 12.

We obtain lower bounds for TSO and PSO by constructing a similar reduction from 3-SAT to TSO and PSO-Consistency. To this end, we extend the above reduction by only adding read events that enforce sequential behavior. Intuitively, we can force the FIFO buffers of TSO and PSO to push each issued write to the memory immediately. Then, the above correctness argument still applies. The number of write events does not change and is still linear in  $n + m$ . This yields the following result. Details are given in the full version.

► **Theorem 14.** *TSO and PSO-Consistency cannot be solved in time  $2^{o(k)}$  unless ETH fails.*

## 6 Conclusion

We studied the problem of checking whether an execution of a shared-memory concurrent program is consistent under the intended behavior of the memory, formalized by a memory model. The main finding is a framework which, given a memory model, yields a deterministic consistency algorithm for it. Obtained algorithms run in time  $\mathcal{O}^*(2^k)$ , where  $k$  is the number of writes in the execution. Technically, the framework works on an abstract memory model and can be instantiated by a concrete one. We applied it to obtain  $\mathcal{O}^*(2^k)$ -time consistency algorithms for SC, TSO, PSO, and RMO. This improves on the formerly known  $\mathcal{O}^*(k^k)$ -time algorithms for these models. Furthermore, for SC, TSO, and PSO we have proven that the obtained algorithms are optimal in the fine-grained sense. To this end, we employed the exponential time hypothesis to show that deterministic consistency algorithms for these models cannot run in time  $2^{o(k)}$  unless the ETH fails. Our framework relies on the assumption of data-independence. It is an interesting question, and considered future work, whether one can obtain a similar framework yielding optimal algorithms if the assumption is dropped.

---

### References

- 1 The sparc architecture manual - version 8 and version 9, 1992,1994.
- 2 H. Sinha A. Heddaya. Coherence, non-coherence and local consistency in distributed shared memory for parallel computing. Technical Report BU-CS-92-004, Boston University, 1992.
- 3 P. A. Abdulla, F. Haziza, L. Holík, B. Jonsson, and A. Rezine. An integrated specification and verification technique for highly concurrent data structures. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2013.
- 4 M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures*, page 251–260. ACM, 1993.
- 5 J. Alglave. A formal hierarchy of weak memory models. *Formal Methods Syst. Des.*, 41(2):178–210, 2012.
- 6 J. Alglave, L. Maranget, and M. Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.*, 36(2):7:1–7:74, 2014.
- 7 R. Alur, K. L. McMillan, and D. A. Peled. Model-checking of correctness conditions for concurrent objects. *Inf. Comput.*, 160(1-2):167–188, 2000.
- 8 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18. ACM, 2010.
- 9 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. What’s decidable about weak memory models? In *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 26–46. Springer, 2012.
- 10 R. Biswas and C. Enea. On the complexity of checking transactional consistency. *Proc. ACM Program. Lang.*, 3(OOPSLA):165:1–165:28, 2019.
- 11 A. Bouajjani, C. Enea, R. Guerraoui, and J. Hamza. On verifying causal consistency. In *POPL*, pages 626–638. ACM, 2017.
- 12 A. Bouajjani, C. Enea, and J. Hamza. Verifying eventual consistency of optimistic replication systems. In *POPL*, pages 285–296. ACM, 2014.
- 13 A. Bouajjani, R. Meyer, and E. Möhlmann. Deciding robustness against total store ordering. In *ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 428–440. Springer, 2011.
- 14 G. Calin, E. Derevenetc, R. Majumdar, and R. Meyer. A theory of partitioned global address spaces. In *FSTTCS*, volume 24 of *LIPICs*, pages 127–139. Schloss Dagstuhl, 2013.
- 15 J. F. Cantin, M. H. Lipasti, and J. E. Smith. The complexity of verifying memory coherence and consistency. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):663–671, 2005.
- 16 J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized np-hard problems. *Inf. Comput.*, 201(2):216–231, 2005.

- 17 P. Chini, J. Kolberg, A. Krebs, R. Meyer, and P. Saivasan. On the complexity of bounded context switching. In *ESA*, volume 87 of *LIPICs*, pages 27:1–27:15. Schloss Dagstuhl, 2017.
- 18 P. Chini, R. Meyer, and P. Saivasan. Fine-grained complexity of safety verification. In *TACAS*, volume 10806 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2018.
- 19 P. Chini, R. Meyer, and P. Saivasan. Complexity of liveness in parameterized systems. In *FSTTCS*, volume 150 of *LIPICs*, pages 37:1–37:15. Schloss Dagstuhl, 2019.
- 20 P. Chini and P. Saivasan. A framework for consistency algorithms. *CoRR*, abs/2007.11398, 2020.
- 21 M. Cygan, H. Dell, D. Lokshantov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.
- 22 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 23 E. Derevenetc and R. Meyer. Robustness against power is pspace-complete. In *ICALP*, volume 8573 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2014.
- 24 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.
- 25 M. Emmi and C. Enea. Monitoring weak consistency. In *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2018.
- 26 M. Emmi and C. Enea. Sound, complete, and tractable linearizability monitoring for concurrent collections. *Proc. ACM Program. Lang.*, 2(POPL):25:1–25:27, 2018.
- 27 M. Emmi, C. Enea, and J. Hamza. Monitoring refinement via symbolic reasoning. In *PLDI*, pages 260–269. ACM, 2015.
- 28 C. Enea and A. Farzan. On atomicity in presence of non-atomic writes. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 497–514. Springer, 2016.
- 29 A. Farzan and P. Madhusudan. The complexity of predicting atomicity violations. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2009.
- 30 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.
- 31 F. Furbach, R. Meyer, K. Schneider, and M. Senftleben. Memory-model-aware testing: A unified complexity analysis. *ACM Trans. Embedded Comput. Syst.*, 14(4):63:1–63:25, 2015.
- 32 P. B. Gibbons and E. Korach. Testing shared memories. *SIAM J. Comput.*, 26(4):1208–1244, 1997.
- 33 J. R. Goodman. Cache consistency and sequential consistency. Technical Report 1006, University of Wisconsin-Madison, 1991.
- 34 M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.
- 35 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *JCSS*, 62(2):367–375, 2001.
- 36 A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.
- 37 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
- 38 L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.
- 39 D. Lokshantov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *SODA*, pages 760–776. SIAM, 2011.
- 40 S. Mador-Haim, L. Maranget, S. Sarkar, K. Memarian, J. Alglave, S. Owens, R. Alur, M. M. K. Martin, P. Sewell, and D. Williams. An axiomatic memory model for POWER multiprocessors. In *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 495–512. Springer, 2012.
- 41 J. Manson, W. Pugh, and S. V. Adve. The java memory model. In *POPL*, pages 378–391. ACM, 2005.
- 42 U. Mathur, A. Pavlogiannis, and M. Viswanathan. The complexity of dynamic data race prediction. In *LICS*, pages 713–727. ACM, 2020.

- 43 C. H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4):631–653, 1979.
- 44 J. S. Sandberg R. J. Lipton. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, 1988.
- 45 S. Sarkar, P. Sewell, J. Alglave, L. Maranget, and D. Williams. Understanding POWER multiprocessors. In *PLDI*, pages 175–186. ACM, 2011.
- 46 R. C. Steinke and G. J. Nutt. A unified theory of shared memory consistency. *J. ACM*, 51(5):800–849, 2004.
- 47 D. B. Terry, M. Theimer, K. Petersen, A. J. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP*, pages 172–183. ACM, 1995.
- 48 H. Wei, Y. Huang, J. Cao, X. Ma, and J. Lu. Verifying PRAM consistency over read/write traces of data replicas. *CoRR*, abs/1302.5161, 2013.
- 49 P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *POPL*, pages 184–193. ACM, 1986.
- 50 R. Zennou, A. Bouajjani, C. Enea, and M. Erradi. Gradual consistency checking. In *CAV*, volume 11562 of *Lecture Notes in Computer Science*, pages 267–285. Springer, 2019.



# Equivalence of Hidden Markov Models with Continuous Observations

Oscar Darwin 

Department of Computer Science, Oxford University, UK  
<https://www.cs.ox.ac.uk/people/oscar.darwin/>  
oscar.darwin@cs.ox.ac.uk

Stefan Kiefer 

Department of Computer Science, Oxford University, UK  
<https://www.cs.ox.ac.uk/people/stefan.kiefer/>  
stefan.kiefer@cs.ox.ac.uk

---

## Abstract

We consider Hidden Markov Models that emit sequences of observations that are drawn from continuous distributions. For example, such a model may emit a sequence of numbers, each of which is drawn from a uniform distribution, but the support of the uniform distribution depends on the state of the Hidden Markov Model. Such models generalise the more common version where each observation is drawn from a finite alphabet. We prove that one can determine in polynomial time whether two Hidden Markov Models with continuous observations are equivalent.

**2012 ACM Subject Classification** Theory of computation → Random walks and Markov chains; Mathematics of computing → Stochastic processes; Theory of computation → Logic and verification

**Keywords and phrases** Markov chains, equivalence, probabilistic systems, verification

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.43

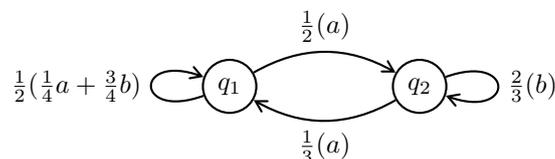
**Related Version** A full version of the paper is available at [12], <https://arxiv.org/abs/2009.12978>.

**Funding** *Oscar Darwin*: Darwin is supported by a Royal Society Enhancement Award.  
*Stefan Kiefer*: Kiefer is supported by a Royal Society University Research Fellowship.

**Acknowledgements** The authors would like to thank anonymous reviewers for their helpful comments and Nikhil Balaji for useful discussions on polynomial identity testing.

## 1 Introduction

A (discrete-time, finite-state) *Hidden Markov Model (HMM)* (often called *labelled Markov chain*) has a finite set  $Q$  of states and for each state a probability distribution over its possible successor states. For any two states  $q, q'$ , whenever the state changes from  $q$  to  $q'$ , the HMM samples and then emits a random observation according to a probability distribution  $D(q, q')$ . For example, consider the following diagram visualising a HMM:



In state  $q_1$ , the successor state is  $q_1$  or  $q_2$ , with probability  $\frac{1}{2}$  each. Upon transitioning from  $q_1$  to itself, observation  $a$  is drawn with probability  $\frac{1}{4}$  and observation  $b$  is drawn with probability  $\frac{3}{4}$ ; upon transitioning from  $q_1$  to  $q_2$ , observation  $a$  is drawn surely.<sup>1</sup>

---

<sup>1</sup> One may allow for observations also on the states and not only on the transitions. But such state observations can be equivalently emitted upon leaving the state. Hence we can assume without loss of generality that all observations are emitted on the transitions.



© Oscar Darwin and Stefan Kiefer;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 43; pp. 43:1–43:14

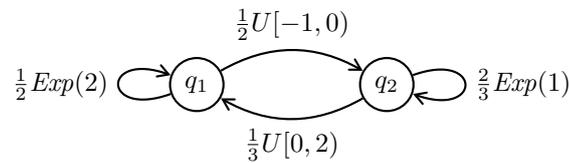


Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this way, a HMM, together with an initial distribution on states, generates a random infinite sequence of observations. In the example above, if the initial distribution is the Dirac distribution on  $q_1$ , the probability that the observation sequence starts with  $a$  is  $\frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2}$  and the probability that the sequence starts with  $ab$  is  $\frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{2}{3}$ .

In the example above the observations are drawn from a finite observation alphabet  $\Sigma = \{a, b\}$ . Indeed, in the literature HMMs most commonly have a finite observation alphabet. In this paper we lift this restriction and consider *continuous-observation* HMMs, by which we mean HMMs as described above, but with continuous observation set  $\Sigma$ . For example, instead of the distributions on  $\{a, b\}$  in the picture above (written there as  $(\frac{1}{4}a + \frac{3}{4}b)$ ,  $(a)$ ,  $(b)$ , respectively), we may have distributions on the real numbers. For example in the following diagram, where  $U[a, b)$  denotes the uniform distribution on  $[a, b)$  and  $Exp(\lambda)$  denotes the exponential distribution with parameter  $\lambda$ :



HMMs, both with finite and infinite observation sets, are widely employed in fields such as speech recognition (see [22] for a tutorial), gesture recognition [7], signal processing [11], and climate modeling [1]. HMMs are heavily used in computational biology [15], more specifically in DNA modeling [9] and biological sequence analysis [14], including protein structure prediction [19] and gene finding [2]. In computer-aided verification, HMMs are the most fundamental model for probabilistic systems; model-checking tools such as Prism [20] and Storm [13] are based on analyzing HMMs efficiently.

One of the most fundamental questions about HMMs is whether two HMMs with initial state distributions are *trace equivalent*, i.e., generate the same distribution on infinite observation sequences. For finite observation alphabets this problem is very well studied and can be solved in polynomial time using algorithms that are based on linear algebra [23, 21, 24, 10]. Checking trace equivalence is used in the verification of obliviousness and anonymity, properties that are hard to formalize in temporal logics, see, e.g., [3, 18, 5].

Although the generalisation to continuous observations (such as passed time, consumed energy, sensor readings) is natural, there has been little work on the algorithmics of such HMMs. One exception is *continuous-time* Markov chains (CTMCs) [4, 8] which are similar to HMMs described above, but with two kinds of observations: on the one hand they emit observations from a finite alphabet, but on the other hand they also emit the *time* spent in each state. Typically, each state-to-state transition is labelled with a parameter  $\lambda$ ; for each transition its time of “firing” is drawn from an exponential distribution with parameter  $\lambda$ ; the transition with the smallest firing time “wins” and causes the corresponding change of state. CTMCs have attractive properties: they are in a sense memoryless, and for many analyses, including model checking, an equivalent discrete-time model can be calculated using an efficient and numerically stable process called *uniformization* [16].

In [17] a stochastic model more general than ours was introduced, allowing not only for uncountable sets of observations (called *labels* there), but also for infinite sets of states and actions. The paper [17] focuses on bisimulation; trace equivalence is not considered. It emphasizes nondeterminism, a feature we do not consider here.

To the best of the authors' knowledge, this paper is the first to study equivalence of HMMs with continuous observations. As continuous functions are part of the input, an equivalence checking algorithm, if it exists (which is not a priori clear), needs to be *symbolic*, i.e., needs to perform computations on functions. Our contributions are as follows:

1. We show in Section 3 that certain aspects of the linear-algebra based approach for checking equivalence of finite-observation HMMs carry over to the continuous case naturally. In particular, equivalence reduces to orthogonality in a certain vector space of state-indexed real vectors, see Proposition 7.
2. However, we show in Section 4 that in the continuous case there can be *additional* linear dependencies between the observation density functions (which is impossible in the finite case, where the different observations can be assumed linearly independent). This renders a simple-minded reduction to the finite case incorrect. Therefore, an equivalence checking algorithm needs to consider the interplay with the vector space from item 1.
3. For the required computations on the observation density functions we introduce in Section 5 *linearly decomposable profile languages*, which are languages (i.e., sets of finite words) whose elements encode density functions on which basis computations can be performed efficiently. In Section 5.1 we provide an extensive example of such a language, encoding (linear combinations of) Gaussian, exponential, and piecewise polynomial density functions. The proof that this language has the required properties is non-trivial itself and requires *alternant matrices* and comparisons of the tails of various density functions.
4. In Section 6 we finally show that HMMs whose observation densities are given in terms of linearly decomposable profile languages can be checked for equivalence in *polynomial time*, by a reduction to the finite-observation case. We also indicate, in Example 23, how our result can be used to check for susceptibility of certain timing attacks.

## 2 Preliminaries

We write  $\mathbb{N}$  for the set of positive integers,  $\mathbb{Q}$  for the set of rationals and  $\mathbb{Q}_+$  for the set of positive rationals. For  $d \in \mathbb{N}$  and a finite set  $Q$  we use the notation  $|Q|$  for the number of elements in  $Q$ ,  $[d] = \{1, \dots, d\}$  and  $[Q] = \{1, \dots, |Q|\}$ . Vectors  $\mu \in \mathbb{R}^N$  are viewed as row vectors and we write  $\mathbb{1} = (1, \dots, 1) \in \mathbb{R}^N$ . Superscript  $T$  denotes transpose; e.g.,  $\mathbb{1}^T$  is a column vector of ones. A matrix  $M \in \mathbb{R}^{N \times N}$  is *stochastic* if  $M$  is non-negative and  $\sum_{j=1}^N M_{i,j} = 1$  for all  $i \in [N]$ . For a domain  $\Sigma$  and subset  $E \subseteq \Sigma$  the *characteristic* function  $\chi_E : \Sigma \rightarrow \{0, 1\}$  is defined as  $\chi_E(x) = 1$  if  $x \in E$  and  $\chi_E(x) = 0$  otherwise.

Throughout this paper, we use  $\Sigma$  to denote a set of *observations*. We assume  $\Sigma$  is a topological space and  $(\Sigma, \mathcal{G}, \lambda)$  is a measure space where all the open subsets of  $\Sigma$  are contained within  $\mathcal{G}$  and have non-zero measure. Indeed  $\mathbb{R}$  and the usual Lebesgue measure space on  $\mathbb{R}$  satisfy these assumptions. The set  $\Sigma^n$  is the set of words over  $\Sigma$  of length  $n$  and  $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ .

A matrix valued function  $\Psi : \Sigma \rightarrow [0, \infty)^{N \times N}$  can be integrated element-wise. We write  $\int_E \Psi d\lambda$  for the matrix with entries  $(\int_E \Psi d\lambda)_{i,j} = \int_E \Psi_{i,j} d\lambda$ , where  $\Psi_{i,j} : \Sigma \rightarrow [0, \infty)$  is defined by  $\Psi_{i,j}(x) = (\Psi(x))_{i,j}$  for all  $x \in \Sigma$ .

A function  $f : \Sigma \rightarrow \mathbb{R}^m$  is *piecewise continuous* if there is an open set  $C \subseteq \Sigma$ , called a *set of continuity*, such that  $f$  is continuous on  $C$  and for every point  $x \in \Sigma \setminus C$  there is some sequence of points  $x_n \in C$  such that  $\lim_{n \rightarrow \infty} x_n = x$  and  $\lim_{n \rightarrow \infty} f(x_n) = f(x)$ . For a non-negative function  $f : \Sigma \rightarrow [0, \infty)$  we use the notation  $\text{supp } f = \{x \in \Sigma \mid f(x) > 0\}$ .

► **Definition 1.** A Hidden Markov Model (HMM) is a triple  $(Q, \Sigma, \Psi)$  where  $Q$  is a finite set of states,  $\Sigma$  is a set of observations, and the observation density matrix  $\Psi : \Sigma \rightarrow [0, \infty)^{|Q| \times |Q|}$  specifies the transitions such that  $\int_{\Sigma} \Psi d\lambda$  is a stochastic matrix.

► **Example 2.** The second HMM from the introduction is the triple  $(\{q_1, q_2\}, \mathbb{R}, \Psi)$  with

$$\Psi(x) = \begin{pmatrix} \frac{1}{2} \cdot 2 \exp(-2x) \cdot \chi_{[0, \infty)}(x) & \frac{1}{2} \cdot 1 \cdot \chi_{[-1, 0)}(x) \\ \frac{1}{3} \cdot \frac{1}{2} \cdot \chi_{[0, 2)}(x) & \frac{2}{3} \cdot \exp(-x) \cdot \chi_{[0, \infty)}(x) \end{pmatrix}. \quad (1)$$

We assume that  $\Psi$  is piecewise continuous and extend  $\Psi$  to the mapping  $\Psi : \Sigma^* \rightarrow [0, \infty)^{|Q| \times |Q|}$  with  $\Psi(x_1 \cdots x_n) = \Psi(x_1) \times \cdots \times \Psi(x_n)$  for  $x_1, \dots, x_n \in \Sigma$ . If  $C$  is the set of continuity for  $\Psi : \Sigma \rightarrow [0, \infty)^{|Q| \times |Q|}$ , then for fixed  $n \in \mathbb{N}$  the restriction  $\Psi : \Sigma^n \rightarrow [0, \infty)^{|Q| \times |Q|}$  is piecewise continuous with set of continuity  $C^n$ . We say that  $A \subseteq \Sigma^n$  is a *cylinder set* if  $A = A_1 \times \cdots \times A_n$  and  $A_i \in \mathcal{G}$  for  $i \in [n]$ . For every  $n$  there is an induced measure space  $(\Sigma^n, \mathcal{G}^n, \lambda^n)$  where  $\mathcal{G}^n$  is the smallest  $\sigma$ -algebra containing all cylinder sets in  $\Sigma^n$  and  $\lambda^n(A_1 \times \cdots \times A_n) = \prod_{i=1}^n \lambda(A_i)$  for any cylinder set  $A_1 \times \cdots \times A_n$ . Let  $A \subseteq \Sigma^n$  and write  $A\Sigma^\omega$  for the set of infinite words over  $\Sigma$  where the first  $n$  observations fall in the set  $A$ . Given a HMM  $(Q, \Sigma, \Psi)$  and initial distribution  $\pi$  on  $Q$  viewed as vector  $\pi \in \mathbb{R}^{|Q|}$ , there is an induced probability space  $(\Sigma^\omega, \mathcal{G}^*, \mathbb{P}_\pi)$  where  $\Sigma^\omega$  is the set of infinite words over  $\Sigma$ , and  $\mathcal{G}^*$  is the smallest  $\sigma$ -algebra containing (for all  $n \in \mathbb{N}$ ) all sets  $A\Sigma^\omega$  where  $A \subseteq \Sigma^n$  is a cylinder set and  $\mathbb{P}_\pi$  is the unique probability measure such that  $\mathbb{P}_\pi(A\Sigma^\omega) = \pi \int_A \Psi d\lambda^n \mathbb{I}^T$  for any cylinder set  $A \subseteq \Sigma^n$ .

► **Definition 3.** For two distributions  $\pi_1$  and  $\pi_2$  and a HMM  $C = (Q, \Sigma, \Psi)$ , we say that  $\pi_1$  and  $\pi_2$  are equivalent, written  $\pi_1 \equiv_C \pi_2$ , if  $\mathbb{P}_{\pi_1}(A) = \mathbb{P}_{\pi_2}(A)$  holds for all measurable subsets  $A \subseteq \Sigma^\omega$ .

One could define equivalence of two pairs  $(C_1, \pi_1)$  and  $(C_2, \pi_2)$  where  $C_i = (Q_i, \Sigma, \Psi_i)$  are HMMs and  $\pi_i$  are initial distributions for  $i = 1, 2$ . We do not need that though, as we can define, in a natural way, a single HMM over the disjoint union of  $Q_1$  and  $Q_2$  and consider instead equivalence of  $\pi_1$  and  $\pi_2$  (where  $\pi_1, \pi_2$  are appropriately padded with zeros).

Given an observation density matrix  $\Psi$ , a *functional decomposition* consists of functions  $f_k : \Sigma \rightarrow [0, \infty)$  and matrices  $P_k \in \mathbb{R}^{|Q| \times |Q|}$  for  $k \in [d]$  such that  $\Psi(x) = \sum_{k=1}^d f_k(x) P_k$  for all  $x \in \Sigma$  and  $\int_\Sigma f_k d\lambda = 1$  for all  $k \in [d]$ . We sometimes abbreviate this decomposition as  $\Psi = \sum_{k=1}^d f_k P_k$  and this notion has a central role in our paper.

► **Example 4.** The observation density matrix  $\Psi$  from Example 2 has a functional decomposition

$$\begin{aligned} \Psi(x) = & 2 \exp(-2x) \chi_{[0, \infty)}(x) \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 0 \end{pmatrix} + \chi_{[-1, 0)}(x) \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{pmatrix} + \\ & \frac{1}{2} \chi_{[0, 2)}(x) \begin{pmatrix} 0 & 0 \\ \frac{1}{3} & 0 \end{pmatrix} + \exp(-x) \chi_{[0, \infty)}(x) \begin{pmatrix} 0 & 0 \\ 0 & \frac{2}{3} \end{pmatrix} \end{aligned}$$

► **Lemma 5.** Let  $(Q, \Sigma, \Psi)$  be a HMM. If  $\Psi$  has functional decomposition  $\Psi = \sum_{k=1}^d f_k P_k$  then  $\sum_{k=1}^d P_k$  is stochastic.

**Proof.** By definition of a HMM,  $\int_\Sigma \Psi d\lambda$  is stochastic, and we have

$$\int_\Sigma \Psi d\lambda = \int_\Sigma \sum_{k=1}^d f_k P_k d\lambda = \sum_{k=1}^d P_k \int_\Sigma f_k d\lambda = \sum_{k=1}^d P_k. \quad \blacktriangleleft$$

When  $\Sigma$  is finite, it follows that  $\int_\Sigma \Psi d\lambda = \sum_{a \in \Sigma} \Psi(a)$ . Hence  $\sum_{a \in \Sigma} \Psi(a)$  is stochastic.

**Encoding.** For computational purposes we assume that rational numbers are represented as ratios of integers in binary. The initial distribution of a HMM with state set  $Q$  is given as a vector  $\pi \in \mathbb{Q}^{|Q|}$ . We also need to encode continuous functions, in particular, density functions such as Gaussian, exponential or piecewise-polynomial functions. A *profile* is a finite word (i.e., string) that describes a continuous function. It may consist of (an encoding of) a function type and its parameters. For example, the profile  $(\mathcal{N}, \mu, \sigma)$  may denote a Gaussian (also called normal) distribution with mean  $\mu \in \mathbb{Q}$  and standard deviation  $\sigma \in \mathbb{Q}_+$ . A profile may also consist of a description of a rational linear combination of such building blocks. For any profile  $\gamma$  we write  $\llbracket \gamma \rrbracket : \Sigma \rightarrow [0, \infty)$  for the function it encodes. For example, a profile  $\gamma = (\mathcal{N}, \mu, \sigma)$  with  $\mu \in \mathbb{Q}$ ,  $\sigma \in \mathbb{Q}_+$  may encode the function  $\llbracket \gamma \rrbracket : \mathbb{R} \rightarrow [0, \infty)$  given as  $\llbracket \gamma \rrbracket(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp -\frac{(x-\mu)^2}{2\sigma^2}$ . Without restricting ourselves to any particular encoding, we assume that  $\Gamma$  is a *profile language*, i.e., a finitely presented but usually infinite set of valid profiles. For any  $\Gamma_0 \subseteq \Gamma$  we write  $\llbracket \Gamma_0 \rrbracket = \{\llbracket \gamma \rrbracket \mid \gamma \in \Gamma_0\}$ .

We use profiles to encode HMMs  $C = (Q, \Sigma, \Psi)$ : we say that  $C$  is *over*  $\Gamma$  if the observation density matrix  $\Psi$  is given as a matrix of pairs  $(p_{i,j}, \gamma_{i,j}) \in \mathbb{Q}_+ \times \Gamma$  such that  $\Psi_{i,j} = p_{i,j} \llbracket \gamma_{i,j} \rrbracket$  and  $\int_{\Sigma} \llbracket \gamma_{i,j} \rrbracket d\lambda = 1$  hold for all  $i, j \in [Q]$ . In this way the  $p_{i,j}$  form the transition probabilities between states and the  $\gamma_{i,j}$  encode the probability densities of the observations upon each transition.

► **Example 6.** For a suitable profile language  $\Gamma$ , the HMM from Example 2 may be over  $\Gamma$ , with the observation density matrix given as

$$\begin{pmatrix} (\frac{1}{2}, (Exp, 2)) & (\frac{1}{2}, (U, -1, 0)) \\ (\frac{1}{3}, (U, 0, 2)) & (\frac{2}{3}, (Exp, 1)) \end{pmatrix} \quad (2)$$

The observation density matrix  $\Psi$  of a HMM  $(Q, \Sigma, \Psi)$  with *finite*  $\Sigma$  can be given as a list of matrices  $\Psi(a) \in \mathbb{Q}_+^{|Q| \times |Q|}$  for all  $a \in \Sigma$  such that  $\sum_{a \in \Sigma} \Psi(a)$  is a stochastic matrix.

### 3 Equivalence as Orthogonality

For finite-observation HMMs it is well known [23, 21, 24, 10] that two initial distributions given as vectors  $\pi_1, \pi_2 \in \mathbb{R}^{|Q|}$  are equivalent if and only if  $\pi_1 - \pi_2$  is orthogonal (written as  $\perp$ ) to a certain vector space. Indeed, this property holds more generally:

► **Proposition 7.** *Consider a HMM  $(Q, \Sigma, \Psi)$ . For any  $\pi_1, \pi_2 \in \mathbb{R}^{|Q|}$  we have*

$$\pi_1 \equiv \pi_2 \iff \pi_1 - \pi_2 \perp \text{span} \{ \Psi(w) \mathbb{I}^T \mid w \in \Sigma^* \}.$$

The general case is proven in [12]. In the finite-observation case, Proposition 7 leads to an efficient algorithm for deciding equivalence: it suffices to compute a basis for  $\mathcal{V} = \text{span} \{ \Psi(w) \mathbb{I}^T \mid w \in \Sigma^* \}$ . This can be done using a fixed-point algorithm that computes a sequence of (bases of) increasing subspaces of  $\mathcal{V}$ : start with  $\mathcal{B} = \{ \mathbb{I}^T \}$ , and as long as there is  $a \in \Sigma$  and  $v \in \mathcal{B}$  such that  $\Psi(a)v \notin \text{span } \mathcal{B}$ , add  $\Psi(a)v$  to  $\mathcal{B}$ . Since  $\dim \mathcal{V} \leq |Q|$ , this algorithm terminates after at most  $|Q|$  iterations, and returns  $\mathcal{B}$  such that  $\text{span } \mathcal{B} = \mathcal{V}$ . It is then easy to check whether  $\pi_1 - \pi_2 \perp \mathcal{V}$ . It follows:

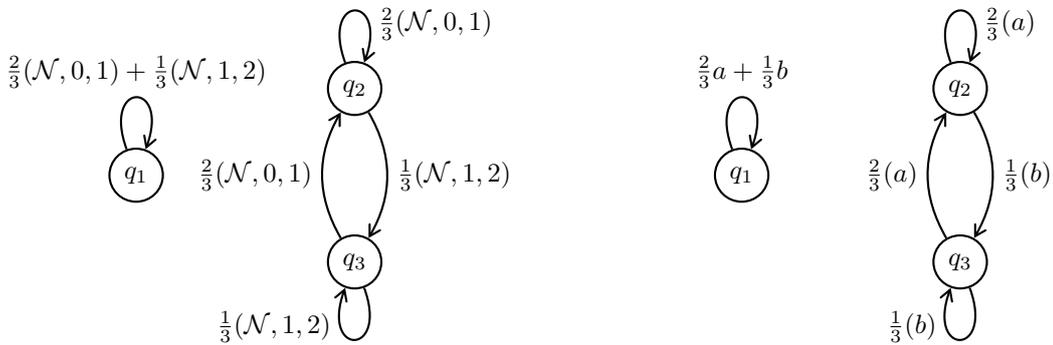
► **Proposition 8.** *Given a HMM  $(Q, \Sigma, \Psi)$  with finite  $\Sigma$  and initial distributions  $\pi_1, \pi_2 \in \mathbb{Q}^{|Q|}$ , it is decidable in polynomial time whether  $\pi_1 \equiv \pi_2$ .*

This is not an effective algorithm when  $\Sigma$  is infinite.

**4** Labelling Reductions

Our goal is to reduce in polynomial time the equivalence problem in continuous-observation HMMs to the equivalence problem in finite-observation HMMs. Since the latter is decidable in polynomial time by Proposition 8, a polynomial time algorithm for deciding equivalence in continuous-observation HMMs follows.

Towards this objective, consider a reduction where each continuous density function is given a label and these labels form the observation alphabet of a finite-observation HMM. For example consider the chain on the left in the diagram below. This disconnected HMM emits letters from two distinct normal distributions with profiles  $(\mathcal{N}, 0, 1)$  and  $(\mathcal{N}, 1, 2)$ . Assigning each distribution letters  $a, b$  respectively yields the HMM given on the right. Since in the right chain states  $q_1$  and  $q_2$  are equivalent so too are the same labelled states in the continuous chain.



More rigorously, if  $C = (Q, \Sigma, \Psi)$  is a HMM over  $\Gamma = \{\beta_1, \dots, \beta_K\}$  and  $\Psi$  is encoded as a matrix of coefficient-profile pairs  $(p_{i,j}, \gamma_{i,j}) \in \mathbb{Q}_+ \times \Gamma$  then we call the *labelling reduction* the HMM  $(Q, \hat{\Sigma}, \hat{M})$  where  $\hat{\Sigma} = \{a_1, \dots, a_K\}$  is an alphabet of fresh observations and

$$\hat{M}_{i,j}(a_k) = \begin{cases} p_{i,j} & \gamma_{i,j} = \beta_k \\ 0 & \text{otherwise.} \end{cases}$$

Since  $\Psi$  has functional decomposition  $\Psi = \sum_{k=1}^K \llbracket \beta_k \rrbracket \hat{M}(a_k)$ , it follows by Lemma 5 that  $\sum_{k=1}^K \hat{M}(a_k)$  is stochastic and the labelling reduction is a well defined HMM which may be computed in polynomial time. As discussed in the previous example, equivalence in the labelling reduction implies equivalence in the original chain:

► **Proposition 9.** *Let  $C = (Q, \Sigma, \Psi)$  be a HMM with labelling reduction  $L = (Q, \hat{\Sigma}, \hat{M})$ . Then for any initial distributions  $\pi_1$  and  $\pi_2$*

$$\pi_1 \equiv_L \pi_2 \implies \pi_1 \equiv_C \pi_2.$$

For the proof of Proposition 9 we use the following lemma proven in [12] which will be re-used in Section 6.

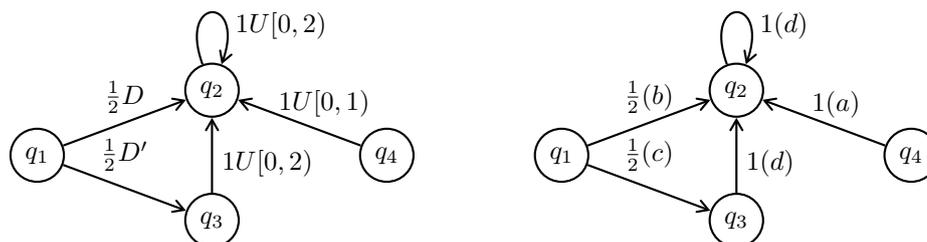
► **Lemma 10.** *Let  $C_1 = (Q, \Sigma_1, \Psi_1)$  and  $C_2 = (Q, \Sigma_2, \Psi_2)$  be two HMMs with the same state space  $Q$ . Suppose that  $\text{span} \{\Psi_1(x) \mid x \in \Sigma_1\} \subseteq \text{span} \{\Psi_2(x) \mid x \in \Sigma_2\}$ . Then, for any two initial distributions  $\pi_1$  and  $\pi_2$ ,*

$$\pi_1 \equiv_{C_2} \pi_2 \implies \pi_1 \equiv_{C_1} \pi_2.$$

**Proof of Proposition 9.**  $\Psi$  has a functional decomposition  $\Psi = \sum_{k=1}^K \llbracket \beta_k \rrbracket \hat{M}(a_k)$ . Thus,  $\text{span} \{ \Psi(x) \mid x \in \Sigma \} \subseteq \text{span} \{ \hat{M}(a_k) \mid a_k \in \hat{\Sigma} \}$  and the statement follows by Lemma 10. ◀

► **Example 11.** Consider the HMMs in the diagram below. The HMM on the left is a continuous-observation chain where  $D$  and  $D'$  are distributions on  $[0, 1]$  with probability density functions  $2x\chi_{[0,1]}(x)$  and  $2(1-x)\chi_{[0,1]}(x)$  respectively, and  $U[a, b]$  is the uniform distribution on  $[a, b]$ . The HMM on the right is the corresponding labelling reduction.

Since  $U[0, 1] = \frac{1}{2}D + \frac{1}{2}D'$ , (the Dirac distributions on) states  $q_1$  and  $q_4$  are equivalent but as the distributions  $U[0, 1], D, D'$  are distinct, they get assigned different labels  $a, b, c$ , respectively in the labelling reduction. The states  $q_1$  and  $q_4$  are therefore not equivalent in the right chain.



## 5 Linearly Decomposable Profile Languages

Example 11 shows that the linear combination of two continuous distributions can “imitate” a single distribution. Therefore we consider the transition densities as part of a vector space of functions. In the usual way  $\mathcal{L}_1(\Sigma, \lambda)$  is the quotient vector space where functions that differ only on a  $\lambda$ -null set are identified. In particular, when  $\Sigma \subseteq \mathbb{R}$  and  $\lambda$  is the Lebesgue measure  $\lambda_{Leb}$ , the functions  $\chi_{[a,b]}$  and  $\chi_{(a,b]}$  are considered the same.

Let  $\Gamma$  be a profile language with  $\llbracket \Gamma \rrbracket \subseteq \mathcal{L}_1(\Sigma, \lambda)$ . We say that  $\Gamma$  is *linearly decomposable* if for every finite set  $\{\gamma_1, \dots, \gamma_n\} = \Gamma_0 \subseteq \Gamma$  one can compute in polynomial time profiles  $\beta_1, \dots, \beta_m \in \Gamma_0$  such that  $\{\llbracket \beta_1 \rrbracket, \dots, \llbracket \beta_m \rrbracket\}$  is a basis for  $\text{span} \{\llbracket \gamma_1 \rrbracket, \dots, \llbracket \gamma_n \rrbracket\}$  (hence  $m \leq n$ ), and further a set of coefficients  $b_{i,j} \in \mathbb{Q}$  for  $i \in [n], j \in [m]$  such that

$$\llbracket \gamma_i \rrbracket = \sum_{j=1}^m b_{i,j} \llbracket \beta_j \rrbracket \text{ for all } i \in [n].$$

The following theorem is the main result of this paper:

► **Theorem 12.** *Given a HMM  $(Q, \Sigma, \Psi)$  over a linearly decomposable profile language, and initial distributions  $\pi_1, \pi_2 \in \mathbb{Q}^{|Q|}$ , it is decidable in polynomial time (in the size of the encoding) whether  $\pi_1 \equiv \pi_2$ .*

We prove Theorem 12 in Section 6. To make the notion of linearly decomposable profile languages more concrete, we give a concrete example in the following subsection.

### 5.1 Example: Gaussian, Exponential, and Piecewise Polynomial Functions

We describe a profile language,  $\Gamma_{GEM}$ , that can specify linear combinations of Gaussian, exponential, and piecewise polynomial density functions.

We call a function of the form  $x \mapsto x^k \chi_I(x)$  where  $k \in \mathbb{N} \cup \{0\}$  and  $I \subset \mathbb{R}$  is an interval an *interval-domain monomial*. To avoid clutter, we often denote interval-domain monomials only by  $x^k \chi_I$ . Recall that  $\mathcal{L}_1(\mathbb{R}, \lambda_{Leb})$  is a quotient space, so half open intervals  $I = [a, b)$  are sufficient. Any piecewise polynomial is a linear combination of interval-domain monomials.

Let  $M$  be a set of profiles encoding interval-domain monomials  $x^k \chi_{[a,b)}$  in terms of  $k \in \mathbb{N} \cup \{0\}$  and  $a, b \in \mathbb{Q}$ . Gaussian and exponential density functions can be fully described using their parameters, which we assume to be rational. We write  $G$  and  $E$  for corresponding sets of profiles, respectively. Finally, we fix a profile language  $\Gamma_{GEM} \supset G \cup E \cup M$  obtained by closing  $G \cup E \cup M$  under linear combinations. That is, for any  $\gamma_1, \dots, \gamma_k \in \Gamma_{GEM}$  and  $\lambda_1, \dots, \lambda_k \in \mathbb{Q}$ , there exists a profile  $\gamma \in \Gamma_{GEM}$  such that  $\llbracket \gamma \rrbracket = \lambda_1 \llbracket \gamma_1 \rrbracket + \dots + \lambda_k \llbracket \gamma_k \rrbracket$ . This closure can be achieved using a specific constructor, say  $\mathcal{S}$ , for linear combinations, so that  $\gamma = \mathcal{S}(\lambda_1, \gamma_1, \dots, \lambda_k, \gamma_k)$ .

► **Example 13.** The HMM  $(Q, \mathbb{R}, \Psi)$  from Example 11 is over  $\Gamma_{GEM}$ : the observation density matrix  $\Psi$  can be encoded as a matrix of coefficient-profile pairs

$$\begin{pmatrix} 0 & (\frac{1}{2}, \gamma_1) & (\frac{1}{2}, \gamma_2) & 0 \\ 0 & (1, \gamma_3) & 0 & 0 \\ 0 & (1, \gamma_3) & 0 & 0 \\ 0 & (1, \gamma_4) & 0 & 0 \end{pmatrix}$$

with  $\gamma_1, \gamma_2, \gamma_3, \gamma_4 \in \Gamma_{GEM}$  and  $\llbracket \gamma_1 \rrbracket = 2x\chi_{[0,1)}$  and  $\llbracket \gamma_2 \rrbracket = 2(1-x)\chi_{[0,1)}$  and  $\llbracket \gamma_3 \rrbracket = \frac{1}{2}\chi_{[0,2)}$  and  $\llbracket \gamma_4 \rrbracket = \chi_{[0,1)}$ .

► **Lemma 14.** *Let  $H$  be a set of disjoint half open intervals. Suppose that  $m_1, \dots, m_I$  are distinct interval-domain monomials such that  $\text{supp } m_i \in H$  for all  $i \in [I]$ . In addition, let  $g_1, \dots, g_J$  and  $e_1, \dots, e_K$  be distinct Gaussian and exponential density functions, respectively. Then, the set  $\{m_1, \dots, m_I, g_1, \dots, g_J, e_1, \dots, e_K\}$  is linearly independent.*

For the proof of this lemma we need a result concerning *alternant matrices*. Consider functions  $f_1, \dots, f_n : \Sigma \rightarrow \mathbb{R}$  and let  $x_1, \dots, x_n \in \Sigma$ . Then,

$$M = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_n(x_n) \end{pmatrix}$$

is called the alternant matrix for  $f_1, \dots, f_n$  and *input points*  $x_1, \dots, x_n$ .

► **Lemma 15.** *Suppose  $f_1, \dots, f_n \in \mathcal{L}_1(\Sigma, \lambda)$ . Then, the  $f_i$  are linearly dependent if and only if all alternant matrices for the  $f_i$  are singular.*

**Sketch proof of Lemma 14.** Under the assumption that a linear combination exists almost surely equal to 0, by examining the limit at  $+\infty$  we show that the exponential and Gaussian coefficients are zero. Then, by constructing an appropriate alternant matrix with full rank we invoke Lemma 15 which means the remaining interval-domain monomials are linearly independent and thus must also have zero coefficients. ◀

► **Proposition 16.** *The profile language  $\Gamma_{GEM}$  is linearly decomposable.*

Full proofs of Lemmas 14 and 15 and Proposition 16 can be found in [12]. From the latter we obtain the following corollary of Theorem 12:

► **Corollary 17.** *Given a HMM  $(Q, \Sigma, \Psi)$  over  $\Gamma_{GEM}$ , and initial distributions  $\pi_1, \pi_2 \in \mathbb{Q}^{|Q|}$ , it is decidable in polynomial time whether  $\pi_1 \equiv \pi_2$ .*

**6 Proof of Theorem 12**

Suppose that  $\Psi$  has a functional decomposition  $\sum_{k=1}^d f_k P_k$  such that the set  $\{f_1, \dots, f_d\}$  is linearly independent. Then,  $\sum_{k=1}^d f_k P_k$  is called an *independent functional decomposition*. The efficient computation of an independent functional decomposition is the key ingredient for the proof of Theorem 12. We start with the following lemma.

► **Lemma 18.** *Suppose  $\Psi : \Sigma \rightarrow [0, \infty)^{|Q| \times |Q|}$  has an independent functional decomposition  $\Psi = \sum_{k=1}^d f_k P_k$ . Then,  $\text{span} \{\Psi(x) \mid x \in \Sigma\} = \text{span} \{P_k \mid k \in [d]\}$ .*

**Proof.** Since  $\Psi(x) = \sum_{k=1}^d f_k(x) P_k$ , we have  $\text{span} \{\Psi(x) \mid x \in \Sigma\} \subseteq \text{span} \{P_k \mid k \in [d]\}$ . For the reverse inclusion, since the  $f_i$  are linearly independent, by Lemma 15 there exists an alternant matrix  $M$  with full rank for  $f_1, \dots, f_d$  with input points  $x_1, \dots, x_d$ . Hence, for each of the standard basis vectors  $e_k \in \{0, 1\}^d$ ,  $k \in [d]$ , there exists  $v_k = (v_{1,k}, \dots, v_{d,k}) \in \mathbb{R}^d$  such that  $v_k M = e_k$ . Writing  $\delta_{j,k}$  for the Kronecker delta function it follows that

$$\sum_{i=1}^d v_{i,k} \Psi(x_i) = \sum_{i=1}^d v_{i,k} \sum_{j=1}^d f_j(x_i) P_j = \sum_{j=1}^d P_j \sum_{i=1}^d v_{i,k} f_j(x_i) = \sum_{j=1}^d P_j \delta_{j,k} = P_k,$$

which implies that  $\text{span} \{\Psi(x) \mid x \in \Sigma\} \supseteq \text{span} \{P_k \mid k \in [d]\}$ . ◀

The proof of the following proposition re-uses Lemma 10 from Section 4.

► **Proposition 19.** *Suppose that HMM  $C = (Q, \Sigma, \Psi)$  has independent functional decomposition  $\Psi = \sum_{k=1}^d f_k P_k$  and each  $P_k$  is non-negative for all  $k \in [d]$ . Define a set  $\bar{\Sigma} = \{a_1, \dots, a_d\}$  of fresh observations and the observation density matrix  $M$  with  $M(a_k) = P_k$  for all  $k \in [d]$ . Then  $F = (Q, \bar{\Sigma}, M)$  is a finite-observation HMM and for any initial distributions  $\pi_1, \pi_2$*

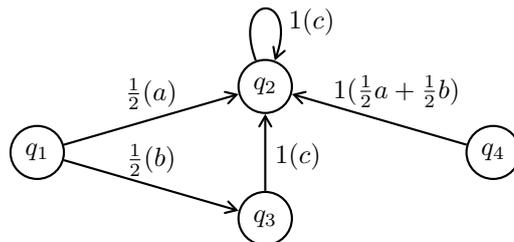
$$\pi_1 \equiv_C \pi_2 \iff \pi_1 \equiv_F \pi_2.$$

**Proof.** It follows by Lemma 5 that  $\sum_{k=1}^d P_k$  is stochastic. Thus  $F$  defines a HMM. By Lemma 18,  $\text{span} \{\Psi(x) \mathbb{I}^T \mid x \in \Sigma\} = \text{span} \{M(a) \mathbb{I}^T \mid a \in \bar{\Sigma}\}$  which combined with Lemma 10 gives the result. ◀

► **Example 20.** We use the HMM  $C$  discussed in Examples 11 and 13 to illustrate the construction of Proposition 19. The basis  $\{2x\chi_{[0,1)}, 2(1-x)\chi_{[0,1)}, \frac{1}{2}\chi_{[0,2)}\}$  leads to the independent functional decomposition

$$\Psi = 2x\chi_{[0,1)} \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \end{pmatrix} + 2(1-x)\chi_{[0,1)} \begin{pmatrix} 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \end{pmatrix} + \frac{1}{2}\chi_{[0,2)} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Therefore, Proposition 19 implies that two initial distributions  $\pi_1, \pi_2 \in \mathbb{R}^{|Q|}$  are equivalent in  $C$  if and only if they are equivalent in the following HMM:



Here, states  $q_1$  and  $q_4$  are equivalent. Hence, they are also equivalent in  $C$ .

## 43:10 Equivalence of Hidden Markov Models with Continuous Observations

If an observation density matrix has an entry with pdf  $2e^{-x} - 2e^{-2x}$  (which is encodable in  $\Gamma_{GEM}$  due to its convex closure property), the independent functional decomposition generated by the algorithm described in the proof of Proposition 16 in [12] has matrices which are not all non-negative. Therefore, Proposition 19 cannot be applied directly. However, given an independent functional decomposition  $\Psi = \sum_{k=1}^d f_k P_k$  and noting that  $\sum_{k=1}^d P_k$  is stochastic by Lemma 5, the following proposition shows that there is a small  $\theta > 0$  such that  $P - \theta P_k$  is non-negative for all  $k \in [d]$ . Furthermore,  $\text{span}\{P_k \mid k \in [d]\} = \text{span}\{P - \theta P_k \mid k \in [d]\}$ . These two facts lead us to construct a finite-observation HMM using the scaled transition matrices  $\frac{1}{d-\theta}(P - \theta P_k)$ .

► **Proposition 21.** *Let  $C = (Q, \Sigma, \Psi)$  be a HMM with independent functional decomposition  $\Psi = \sum_{k=1}^d f_k P_k$ . Let  $P = \sum_{k=1}^d P_k$  and*

$$\theta = \min \left\{ \frac{1}{2}, \frac{\min\{(P)_{i,j} \mid (P)_{i,j} > 0\}}{\max\{(P_k)_{i,j} \mid i, j \in [Q], k \in [d]\}} \right\}.$$

Define an alphabet  $\tilde{\Sigma} = \{a_1, \dots, a_d\}$  of fresh observations and the HMM  $F = (Q, \tilde{\Sigma}, M)$  with  $M(a_k) = \frac{1}{d-\theta}(P - \theta P_k)$ . Then, for any initial distributions  $\mu_1, \mu_2$

$$\mu_1 \equiv_F \mu_2 \iff \mu_1 \equiv_C \mu_2.$$

**Proof.** First we show that  $F$  is a well-defined HMM. Matrix  $\sum_{k=1}^d M(a_k)$  is stochastic as

$$\sum_{k=1}^d M(a_k) = \frac{1}{d-\theta} \sum_{k=1}^d (P - \theta P_k) = \frac{dP - \theta \sum_{k=1}^d P_k}{d-\theta} = P, \quad (3)$$

and by Lemma 5,  $P$  is stochastic. In addition we must show that  $M(a_k)$  is non-negative for each  $k \in [d]$ . Since  $\theta \leq \frac{1}{2}$ , it is enough to show that  $P - \theta P_k$  is non-negative for each  $k \in [d]$ . Suppose that  $(P)_{i,j} = 0$ . Then,  $\int_{\Sigma} \Psi_{i,j} d\lambda = (P)_{i,j} = 0$ , which implies that  $\Psi_{i,j} = 0$  since  $\Psi$  is piecewise continuous. Thus,  $\sum_{k=1}^d f_k (P_k)_{i,j} = \Psi_{i,j} = 0$ . Since  $\{f_k\}_{k=1}^d$  is linearly independent, it follows that  $(P_k)_{i,j} = 0$  for all  $k \in [d]$  and so  $(P - \theta P_k)_{i,j} = 0$ . Now suppose that  $(P)_{i,j} > 0$ . By the definition of  $\theta$ , it follows that  $(\theta P_k)_{i,j} \leq (P)_{i,j}$ . Thus,  $F$  is a well defined HMM.

Observe that  $\text{span}\{P - \theta P_k \mid k \in [d]\} \subseteq \text{span}\{P_k \mid k \in [d]\}$ . The opposite inclusion follows from the fact that, by Equation (3), we have  $P \in \text{span}\{P - \theta P_k \mid k = 1, \dots, d\}$ . Thus, by Lemma 18,

$$\text{span}\{M(a) \mid a \in \tilde{\Sigma}\} = \text{span}\{P - \theta P_k \mid k \in [d]\} = \text{span}\{P_k \mid k \in [d]\} = \text{span}\{\Psi(x) \mid x \in \Sigma\}$$

and hence, the proposition follows from Lemma 10. ◀

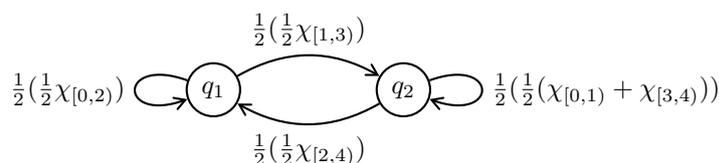
Now we can prove Theorem 12:

**Proof of Theorem 12.** Suppose the HMM  $C = (Q, \Sigma, \Psi)$  is over the linearly decomposable profile language  $\Gamma$ . Let  $\Gamma_0 = \{\gamma_1, \dots, \gamma_n\}$  be the set of profiles appearing in the description of  $\Psi$ . From the description of  $\Psi$  as a matrix of coefficient-profile pairs, we can easily compute matrices  $P'_1, \dots, P'_n \in \mathbb{Q}^{|Q| \times |Q|}$  such that  $\Psi = \sum_{i=1}^n \llbracket \gamma_i \rrbracket P'_i$ . Since  $\Gamma$  is linearly decomposable, one can compute in polynomial time a subset  $\{\beta_1, \dots, \beta_d\} \subseteq \Gamma_0$  such that  $\llbracket \{\beta_1, \dots, \beta_d\} \rrbracket$  is linearly independent and also a set of coefficients  $b_{i,k}$  such that  $\llbracket \gamma_i \rrbracket = \sum_{k=1}^d b_{i,k} \llbracket \beta_k \rrbracket$  for all  $i \in [n]$ . Hence:

$$\Psi = \sum_{i=1}^n \llbracket \gamma_i \rrbracket P'_i = \sum_{i=1}^n \sum_{k=1}^d \llbracket \beta_k \rrbracket b_{i,k} P'_i = \sum_{k=1}^d \llbracket \beta_k \rrbracket \sum_{i=1}^n b_{i,k} P'_i$$

Setting  $P_k = \sum_{i=1}^n b_{i,k} P'_i$  for all  $k \in [d]$ , we thus obtain the independent functional decomposition  $\Psi = \sum_{k=1}^d \llbracket \beta_k \rrbracket P_k$ . Now it is straightforward to compute the finite-observation HMM  $F$  from Proposition 21 in polynomial time, thus reducing the equivalence problem in  $C$  to the equivalence problem in the finite-observation HMM  $F$ . By Proposition 8 the theorem follows.  $\blacktriangleleft$

► **Example 22.** We illustrate aspects of the proof of Theorem 12 using the HMM:



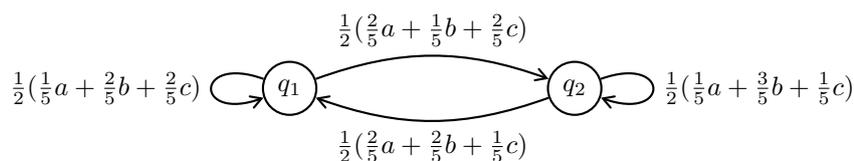
Noting that  $\frac{1}{2}(\chi_{[0,1]} + \chi_{[3,4]}) = \frac{1}{2}\chi_{[0,2]} - \frac{1}{2}\chi_{[1,3]} + \frac{1}{2}\chi_{[2,4]}$  and the set  $\{\frac{1}{2}\chi_{[0,2]}, \frac{1}{2}\chi_{[1,3]}, \frac{1}{2}\chi_{[2,4]}\}$  is linearly independent we obtain the independent functional decomposition

$$\Psi = \frac{1}{2}\chi_{[0,2]} \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} + \frac{1}{2}\chi_{[1,3]} \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} \end{pmatrix} + \frac{1}{2}\chi_{[2,4]} \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

According to Proposition 21,  $P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$ . Further we compute  $\theta = \frac{1}{2}$  and  $d - \theta = \frac{5}{2}$  and

$$\begin{aligned} M(a) &= \frac{2}{5} \left[ \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} - \frac{1}{2} \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \right] = \begin{pmatrix} \frac{1}{10} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{10} \end{pmatrix} \\ M(b) &= \frac{2}{5} \left[ \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} \end{pmatrix} \right] = \begin{pmatrix} \frac{1}{5} & \frac{1}{10} \\ \frac{1}{5} & \frac{3}{10} \end{pmatrix} \\ M(c) &= \frac{2}{5} \left[ \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \right] = \begin{pmatrix} \frac{1}{5} & \frac{1}{5} \\ \frac{1}{10} & \frac{1}{10} \end{pmatrix}. \end{aligned}$$

It follows that any initial distributions  $\pi_1$  and  $\pi_2$  are equivalent in  $(Q, \Sigma, \Psi)$  if and only if they are equivalent in the following HMM:

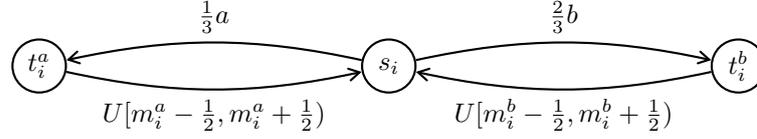


For any initial distributions  $\pi_1, \pi_2 \in \mathbb{Q}^2$  this can be checked with Proposition 8. (In this example  $\pi_1 \equiv \pi_2$  holds only if  $\pi_1 = \pi_2$ .)

► **Example 23.** We also discuss an example, inspired from [6], where HMM non-equivalence means susceptibility to timing attacks, and HMM equivalence means immunity to such attacks. Consider a system that emits two kinds of observations, both visible to an attacker: a function to be executed (we arbitrarily assume a choice between two functions  $a$  and  $b$ , and impute a probability distribution between them) and the time it takes to execute that function. An attacker therefore sees a sequence  $\ell_1 t_1 \ell_2 t_2 \dots$ , where  $\ell_i \in \{a, b\}$  and  $t_i \in [0, \infty)$ . In [6] the times  $t_1, t_2, \dots$  are all identical and depend only on the secret key held by the system, but we assume in the following that the  $t_i$  are drawn from a probability distribution that depends on the function ( $a$  or  $b$ ) and the key. We assume that with key  $i$  the execution

## 43:12 Equivalence of Hidden Markov Models with Continuous Observations

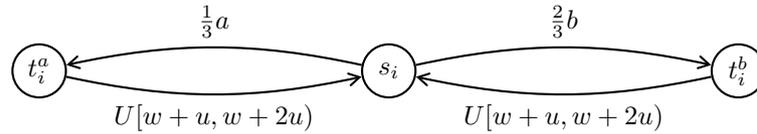
times have uniform distributions  $U[m_i^a - \frac{1}{2}, m_i^a + \frac{1}{2})$  and  $U[m_i^b - \frac{1}{2}, m_i^b + \frac{1}{2})$ . The situation can then be modelled with the HMM below.<sup>2</sup>



A *timing leak* occurs if the attacker can glean the key from the execution times. For example, the attacker can distinguish between keys  $k_1$  and  $k_2$  if and only if states  $s_1$  and  $s_2$  are not equivalent. One can check, using the algorithm we have developed in this section, that  $s_1$  and  $s_2$  are equivalent if and only if  $m_1^a = m_2^a$  and  $m_1^b = m_2^b$ . Moreover, it follows from Section 5 that if instead of  $U[m_1^a - \frac{1}{2}, m_1^a + \frac{1}{2})$  and  $U[m_2^a - \frac{1}{2}, m_2^a + \frac{1}{2})$  we had two distributions with density functions from  $\llbracket \Gamma_{GEM} \rrbracket$  with the same mean and the same variance, states  $s_1, s_2$  would still be non-equivalent whenever the two distributions are not identical.

One may try to guard against this timing leak by “padding” the execution time, so that the sum of the execution time and an added time is constant (and independent of the key). After the execution of the function, an idling loop would be executed until the worst-case (among all keys) execution time of the functions has been reached or exceeded. Let us call this worst-case execution time  $w \in (0, \infty)$ . This idling loop would take time  $u > 0$  in each iteration, so the total idling time is always an integer multiple of  $u$ . It is argued in [6] that this guard is in general ineffective in that the attacker can still glean the execution time modulo  $u$ . Therefore, it is suggested in [6] to add, in addition, a time that is uniformly distributed on  $[0, u)$ .

This remedy also works in our case with random execution times. Indeed, one can show that for any independent random variables  $X, Y$ , where  $Y$  is distributed with  $U[0, u)$ , we have that  $(X + Y) \bmod u$  is distributed with  $U[0, u)$ . Therefore, by adding an independent  $U[0, u)$  random time to the padding described above, the times observable by the attacker now have a  $U[w + u, w + 2u)$  distribution, independent of the key.



All states  $s_i$  are now equivalent, so the key does not leak.

## 7 Conclusions

We have shown that equivalence of continuous-observation HMMs is decidable in polynomial time, by reduction to the finite-observation case. The crucial insight is that, rather than integrating the density functions, one needs to consider them as elements of a vector space and computationally establish linear (in)dependence of functions. Therefore, our polynomial-time reduction performs symbolic computations on continuous density functions. As a suitable framework for these computations we have introduced the notion of linearly decomposable profile languages, and we have established  $\Gamma_{GEM}$  as such a profile language.

<sup>2</sup> In this case the observation set  $\Sigma = [0, \infty) \cup \{a, b\}$  is a disjoint union of topological spaces and there is a natural measure space induced from the Lebesgue measure space on  $[0, \infty)$  and a discrete measure on  $\{a, b\}$ .

In future work, it would be desirable to extend  $\Gamma_{GEM}$  and/or develop other linear decomposable profile languages, including over sets  $\Sigma$  of observations that are not real numbers. The authors believe that the developed computational framework may be the foundation for further algorithms on continuous-observation HMMs. For example, one may want to compute the total-variation distance of two continuous-observation HMMs. Can Markov chains with continuous emissions be model-checked efficiently?

---

## References

- 1 P. Ailliot, C. Thompson, and P. Thomson. Space-time modelling of precipitation by using a hidden Markov model and censored Gaussian distributions. *Journal of the Royal Statistical Society*, 58(3):405–426, 2009.
- 2 M. Alexandersson, S. Cawley, and L. Pachter. SLAM: Cross-species gene finding and alignment with a generalized pair hidden Markov model. *Genome Research*, 13:469–502, 2003.
- 3 M.S. Alvim, M.E. Andrés, C. Palamidessi, and P. van Rossum. Safe equivalences for security properties. In *Theoretical Computer Science*, pages 55–70. Springer, 2010.
- 4 C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.
- 5 M.S. Bauer, R. Chadha, and M. Viswanathan. Modular verification of protocol equivalence in the presence of randomness. In *Computer Security – ESORICS 2017*, pages 187–205. Springer, 2017.
- 6 B.A. Braun, S. Jana, and D. Boneh. Robust and efficient elimination of cache and timing side channels, 2015. [arXiv:1506.00189](https://arxiv.org/abs/1506.00189).
- 7 F.-S. Chen, C.-M. Fu, and C.-L. Huang. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21(8):745–758, 2003.
- 8 T. Chen, M. Diciolla, M.Z. Kwiatkowska, and A. Mereacre. Time-bounded verification of CTMCs against real-time specifications. In *Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 6919 of *LNCS*, pages 26–42. Springer, 2011.
- 9 G.A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.
- 10 C. Cortes, M. Mohri, and A. Rastogi.  $L_p$  distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(04):761–779, 2007.
- 11 M.S. Crouse, R.D. Nowak, and R.G. Baraniuk. Wavelet-based statistical signal processing using hidden Markov models. *IEEE Transactions on Signal Processing*, 46(4):886–902, April 1998.
- 12 Oscar Darwin and Stefan Kiefer. Equivalence of hidden markov models with continuous observations. Technical report, arXiv.org, 2020. Available at <http://arxiv.org/abs/2009.12978>.
- 13 C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Proceedings of Computer Aided Verification (CAV)*, pages 592–600. Springer, 2017.
- 14 R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
- 15 S.R. Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, October 2004.
- 16 W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In *Numerical solution of Markov chains*, pages 357–371, 1991.
- 17 H. Hermanns, J. Krčál, and J. Křetínský. Probabilistic bisimulation: Naturally on distributions. In *CONCUR 2014 – Concurrency Theory*, pages 249–265. Springer, 2014.
- 18 S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 526–540. Springer, 2011.

- 19 A. Krogh, B. Larsson, G. von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580, 2001.
- 20 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- 21 A. Paz. *Introduction to Probabilistic Automata (Computer Science and Applied Mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.
- 22 L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- 23 M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961.
- 24 W.-G. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, April 1992.

# Nivat-Theorem and Logic for Weighted Pushdown Automata on Infinite Words

Manfred Droste 

Institut für Informatik, Universität Leipzig, Germany  
droste@informatik.uni-leipzig.de

Sven Dziadek 

Institut für Informatik, Universität Leipzig, Germany  
dziadek@informatik.uni-leipzig.de

Werner Kuich

Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria  
werner.kuich@tuwien.ac.at

---

## Abstract

Recently, weighted  $\omega$ -pushdown automata have been introduced by Droste, Ésik, Kuich. This new type of automaton has access to a stack and models quantitative aspects of infinite words. Here, we consider a simple version of those automata. The simple  $\omega$ -pushdown automata do not use  $\epsilon$ -transitions and have a very restricted stack access. In previous work, we could show this automaton model to be expressively equivalent to context-free  $\omega$ -languages in the unweighted case. Furthermore, semiring-weighted simple  $\omega$ -pushdown automata recognize all  $\omega$ -algebraic series.

Here, we consider  $\omega$ -valuation monoids as weight structures. As a first result, we prove that for this weight structure and for simple  $\omega$ -pushdown automata, Büchi-acceptance and Muller-acceptance are expressively equivalent. In our second result, we derive a Nivat theorem for these automata stating that the behaviors of weighted  $\omega$ -pushdown automata are precisely the projections of very simple  $\omega$ -series restricted to  $\omega$ -context-free languages. The third result is a weighted logic with the same expressive power as the new automaton model. To prove the equivalence, we use a similar result for weighted nested  $\omega$ -word automata and apply our present result of expressive equivalence of Muller and Büchi acceptance.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic and verification; Theory of computation  $\rightarrow$  Quantitative automata; Theory of computation  $\rightarrow$  Automata over infinite objects; Theory of computation  $\rightarrow$  Grammars and context-free languages

**Keywords and phrases** Weighted automata, Pushdown automata, Infinite words, Weighted logic

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.44

**Funding** *Sven Dziadek*: supported by DFG Research Training Group 1763 (QuantLA).

*Werner Kuich*: partially supported by Austrian Science Fund (FWF): grant no. I1661 N25.

## 1 Introduction

Languages of infinite words or  $\omega$ -languages are intensively researched due to their applications in model checking and verification [30, 3, 9]. Context-free languages of infinite words have been investigated in a fundamental study by Cohen and Gold [10].

Weighted languages allow us to model the use of resources. In formal language theory, we consider a word to be in the language or not. Contrary to this, weighted languages relate words to resources such as costs, gains, probabilities, counts, time, and of course Boolean values. There exist generalizations to several language classes (regular, context-free, star-free languages, etc.), to various structures (words, trees, pictures, nested words, infinite words, etc.) and to different weight structures (semirings, valuation monoids, etc.). See [20] for



© Manfred Droste, Sven Dziadek, and Werner Kuich;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 44; pp. 44:1–44:14



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an overview. While weighted context-free languages already date back to Chomsky and Schützenberger [8], more recently, Droste, Ésik, Kuich [27, 19, 16] generalized context-free languages of infinite words to the weighted setting.

In this paper, we investigate a type of weighted  $\omega$ -pushdown automata called *simple*  $\omega$ -reset pushdown automaton in [12]. They do not allow  $\epsilon$ -transitions and the stack can only be altered by at most one symbol. Simple automata have been shown to be expressively equivalent to general pushdown automata in the unweighted case for finite words [4] and for infinite words [15] (i.e., the language classes accepted by these two kinds of automata coincide). For continuous commutative star-omega semirings we could show in [13, 12, 14] that for every  $\omega$ -algebraic series  $r$ , there exists a simple  $\omega$ -reset pushdown automaton with behavior  $r$ .

Here, we consider  $\omega$ -valuation monoids as weight structures. They include complete semirings but also discounted and average behavior. Valuation monoids first appeared in [22] but their idea is based on [7]. By an example, we show how a basic web server and its average response time for requests can be modeled by a simple  $\omega$ -pushdown automaton with weights in a suitable  $\omega$ -valuation monoid.

Our first main result is the expressive equivalence of Büchi and Muller acceptance for weighted simple  $\omega$ -pushdown automata; i.e., the classes of behaviors of these two weighted automata models coincide.

Then we show several closure properties for weighted  $\omega$ -pushdown automata. Our second main result is a Nivat-like decomposition theorem [31] that shows that by the help of a morphism, we can express the behavior of every weighted  $\omega$ -pushdown automaton as the intersection of an unweighted  $\omega$ -pushdown automaton and a very simple  $\omega$ -series. Nivat's theorem was extended to weighted automata of finite words over semirings by [21].

Büchi, Elgot, Trakhtenbrot [5, 26, 33] (BET-Theorem) proved that regular languages are exactly those languages definable by monadic second-order logic. Their result was extended by Lautemann, Schwentick, Thérien [29] to context-free languages. While both these former results are for finite words, we defined a logic that is expressively equivalent to context-free languages of infinite words (cf. [15]). The BET-Theorem has been extended to the weighted setting [17]. Weighted logics allow the logical description of weights of finite words [17, 24, 34] and also of infinite words [25, 18, 22].

In this paper, as the third main result, we extend the BET-Theorem to weighted simple  $\omega$ -pushdown automata. We extend the logic in [29, 11] and prove its equivalence to weighted simple  $\omega$ -pushdown automata. For the proof, we do not reinvent the wheel but use the already existing BET-Theorem for weighted nested  $\omega$ -word automata [11]. The application of a projection allows us to lift the result on weighted nested  $\omega$ -word automata to weighted simple  $\omega$ -pushdown automata. We show how the quantitative behavior of the basic web server example mentioned above can be described in our weighted matching  $\omega$ -MSO logic.

An expressive equivalence result for arbitrary weighted  $\omega$ -pushdown automata, besides our Nivat-like result, remains open at present.

We structure the paper as follows. We give basic definitions and compare Muller and Büchi acceptance in Section 2. Then, we prove the Nivat-like result in Section 3. Section 4 defines the logic. Section 5 summarizes the known results about weighted nested  $\omega$ -word languages and also shows the new projection. In Section 6, we prove our weighted BET-Theorem.

## 2 Weight Structure and Simple $\omega$ -Pushdown Automata

This section introduces our weight structure, the  $\omega$ -valuation monoids (cf. [22]), and the weighted automata we want to discuss in this paper. At the end of this section, we give our first main result, the comparison of Muller and Büchi acceptance.

An *alphabet* denotes a finite set of symbols. Let  $\mathbb{N}$  be the set of non-negative integers.

A monoid  $(D, +, 0)$  is called *complete*, if it is equipped with sum operations  $\sum_I : D^I \rightarrow D$  for all families  $(a_i \mid i \in I)$  of elements of  $D$ , where  $I$  is an arbitrary index set, such that the following conditions are satisfied:

- (i)  $\sum_{i \in \emptyset} d_i = 0$ ,  $\sum_{i \in \{k\}} d_i = d_k$ ,  $\sum_{i \in \{j,k\}} d_i = d_j + d_k$  for  $j \neq k$ , and
- (ii)  $\sum_{j \in J} \left( \sum_{i \in I_j} d_i \right) = \sum_{i \in I} d_i$  if  $\bigcup_{j \in J} I_j = I$  and  $I_j \cap I_k = \emptyset$  for  $j \neq k$ .

This means that a monoid  $D$  is complete if it has infinitary sum operations (i) that are an extension of the finite sums and (ii) that are associative and commutative (cf. [28]).

For a set  $D$  we denote by  $C \subseteq_{\text{fin}} D$  that  $C$  is a finite subset of  $D$ . Let  $(D_{\text{fin}})^\omega = \bigcup_{C \subseteq_{\text{fin}} D} C^\omega$ . An  $\omega$ -valuation monoid  $(D, +, \text{Val}^\omega, 0)$  consists of a complete monoid  $(D, +, 0)$  and an  $\omega$ -valuation function  $\text{Val}^\omega : (D_{\text{fin}})^\omega \rightarrow D$  such that  $\text{Val}^\omega(d_i)_{i \in \mathbb{N}} = 0$  whenever  $d_i = 0$  for some  $i \in \mathbb{N}$ . A *product  $\omega$ -valuation monoid* ( $\omega$ -pv-monoid) is a tuple  $(D, +, \text{Val}^\omega, \diamond, 0, \mathbb{1})$  where  $(D, +, \text{Val}^\omega, 0)$  is an  $\omega$ -valuation monoid,  $\diamond : D^2 \rightarrow D$  is a product function and further  $\mathbb{1} \in D$ ,  $\text{Val}^\omega(\mathbb{1}^\omega) = \mathbb{1}$  and  $0 \diamond d = d \diamond 0 = 0$ ,  $\mathbb{1} \diamond d = d \diamond \mathbb{1} = d$  for all  $d \in D$ .

A monoid  $(D, +, 0)$  is called *idempotent* if  $d + d = d$  for all  $d \in D$ . An  $\omega$ -valuation monoid  $(D, +, \text{Val}^\omega, 0)$  is equally called *idempotent* if its underlying monoid  $(D, +, 0)$  is idempotent.

In [11, 22],  $\omega$ -valuation monoids are classified by specific properties. More specific  $\omega$ -valuation monoids will later lead to more loose restrictions on our logic. Due to space constraints, we omit properties on  $\omega$ -valuation monoids here and refer the interested reader to [11]. Additionally, we will only present one possible restriction on our logic.

► **Example 1** ( $\omega$ -valuation monoids). The first two examples are inspired by [7].

1. Let  $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$  and  $-\infty + \infty = -\infty$ . Then  $(\bar{\mathbb{R}}, \text{sup}, \text{lim avg}, +, -\infty, 0)$  is an  $\omega$ -pv-monoid where  $\text{lim avg}(d_i)_{i \in \mathbb{N}} = \liminf_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} d_i$ .
2. Let  $\bar{\mathbb{R}}_+ = \{x \in \mathbb{R} \mid x \geq 0\} \cup \{-\infty\}$ . Then  $(\bar{\mathbb{R}}_+, \text{sup}, \text{disc}_\lambda, +, -\infty, 0)$  for  $0 < \lambda < 1$  is an  $\omega$ -pv-monoid where  $\text{disc}_\lambda(d_i)_{i \in \mathbb{N}} = \lim_{n \rightarrow \infty} \sum_{i=0}^n \lambda^i d_i$ .
3. Any complete semiring  $(S, \oplus, \otimes, 0, \mathbb{1})$  is an  $\omega$ -pv-monoid  $(S, \oplus, \otimes, \otimes, 0, \mathbb{1})$ .

As it simplifies the logical characterization, we follow [23, 15] and use a restricted type of pushdown automaton. We call it *simple  $\omega$ -pushdown automaton*. For the unweighted setting, we proved in [15] that this automaton model is expressively equivalent to general  $\omega$ -pushdown automata; for finite words, this equivalence is hidden in [4]. For the weighted case and for continuous semirings, we show a corresponding result for finite words in [13]. For weights in continuous semirings and for infinite words, we showed in [12, 14] that all  $\omega$ -algebraic series are recognized by weighted simple  $\omega$ -pushdown automata.

Simple  $\omega$ -pushdown automata are realtime, i.e. they do not use  $\epsilon$ -transitions. Additionally, we restrict transitions in a way to only allow either to keep the stack unaltered, to push one symbol or to pop one symbol. Thus, let  $\mathcal{S}(\Gamma) = (\{\downarrow\} \times \Gamma) \cup \{\#\} \cup (\{\uparrow\} \times \Gamma)$  be the set of *stack commands* for a stack alphabet  $\Gamma$ . Note that this implies that the automaton can only read the top of the stack when popping it. Additionally, for technical reasons, we start runs with an empty stack and therefore allow to push onto the empty stack.

► **Definition 2.** An (unweighted)  $\omega$ -pushdown automaton ( $\omega$ PDA) over the alphabet  $\Sigma$  is a tuple  $M = (Q, \Gamma, T, I, F)$  where

- $Q$  is a finite set of states,
- $\Gamma$  is a finite stack alphabet,
- $T \subseteq Q \times \Sigma \times Q \times \mathcal{S}(\Gamma)$  is a set of transitions,
- $I \subseteq Q$  is the set of initial states,
- $F \subseteq Q$  is a set of (Büchi-accepting) final states.

► **Definition 3.** A weighted  $\omega$ -pushdown automaton ( $\omega$ WPDA) over the alphabet  $\Sigma$  and the  $\omega$ -valuation monoid  $(D, +, \text{Val}^\omega, 0)$  is a tuple  $M = (Q, \Gamma, T, I, F, \text{wt})$  where

- $(Q, \Gamma, T, I, F)$  is an unweighted  $\omega$ -pushdown automaton over  $\Sigma$ ,
- $\text{wt}: T \rightarrow D$  is a weight function.

► **Definition 4.** A Muller-accepting  $\omega$ -pushdown automaton over the alphabet  $\Sigma$  is a tuple  $M = (Q, \Gamma, T, I, \mathcal{F})$  where  $Q, \Gamma, T, I$  are defined as for  $\omega$ PDA, but  $\mathcal{F} \subseteq 2^Q$  is a set of Muller-accepting subsets of  $Q$ . Similarly, a weighted Muller-accepting  $\omega$ -pushdown automaton over the alphabet  $\Sigma$  and the  $\omega$ -valuation monoid  $D$  is a tuple  $M = (Q, \Gamma, T, I, \mathcal{F}, \text{wt})$ .

A configuration of an  $\omega$ PDA or  $\omega$ WPDA is a pair  $(q, \gamma)$ , where  $q \in Q$  and  $\gamma \in \Gamma^*$ . We define the transition relation between configurations as follows. Let  $\gamma \in \Gamma^*$  and  $t \in T$ . For  $t = (q, a, q', (\downarrow, A))$ , we write  $(q, \gamma) \vdash_M^t (q', A\gamma)$ . For  $t = (q, a, q', \#)$ , we write  $(q, \gamma) \vdash_M^t (q', \gamma)$ . Finally, for  $t = (q, a, q', (\uparrow, A))$ , we write  $(q, A\gamma) \vdash_M^t (q', \gamma)$ . These three types of transitions are called *push*, *internal* and *pop* transitions, respectively.

We denote by  $\text{label}(q, a, q', s) = a$  the *label* and by  $\text{state}(q, a, q', s) = q$  the *state* of a transition. Both, as well as the function  $\text{wt}$  will be extended to infinite sequences of transitions by letting  $\text{label}((t_i)_{i \geq 0}) = (\text{label}(t_i))_{i \geq 0} \in \Sigma^\omega$  for the infinite word constructed from the labels and similar for  $\text{state}((t_i)_{i \geq 0}) \in Q^\omega$  and for  $\text{wt}((t_i)_{i \geq 0}) \in D^\omega$ .

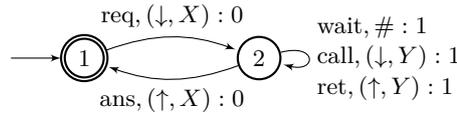
An infinite sequence of transitions  $\rho = (t_i)_{i \geq 0}$  with  $t_i \in T$  is called a *run* of the  $\omega$ WPDA or  $\omega$ PDA  $M$  on  $w = \text{label}(\rho)$  iff there exists an infinite sequence of configurations  $(p_i, \gamma_i)_{i \geq 0}$  with  $p_0 \in I$  and  $\gamma_0 = \epsilon$  such that  $(p_i, \gamma_i) \vdash_M^{t_i} (p_{i+1}, \gamma_{i+1})$  for each  $i \geq 0$ . We abbreviate a run  $\rho = (t_i)_{i \geq 0}$  with  $(p_0, \gamma_0) \vdash_M^{t_0} (p_1, \gamma_1) \vdash_M^{t_1} \dots$  where  $\text{label}(t_i) = a_i$  by  $\rho: (p_0, \gamma_0) \xrightarrow{a_0} (p_1, \gamma_1) \xrightarrow{a_1} \dots$  such that the word becomes visible.

For an infinite sequence of states  $(q_i)_{i \geq 0}$ , let  $\text{Inf}((q_i)_{i \geq 0}) = \{q \mid q = q_i \text{ for infinitely many } i \geq 0\}$  be the set of states that occur infinitely often. For Büchi-accepting automata, a run  $\rho$  is called *successful* if  $\text{Inf}(\text{state}(\rho)) \cap F \neq \emptyset$ . For Muller-accepting automata, a run  $\rho$  is called *successful* if  $\text{Inf}(\text{state}(\rho)) \in \mathcal{F}$ . For an  $\omega$ PDA  $M = (Q, \Gamma, T, I, F)$ , the language *accepted* by  $M$  is denoted by  $\mathcal{L}(M) = \{w \in \Sigma^\omega \mid \exists \text{ successful run of } M \text{ on } w\}$ . A language  $L \subseteq \Sigma^\omega$  is called  $\omega$ PDA-*recognizable* if there exists an  $\omega$ PDA  $M$  with  $\mathcal{L}(M) = L$ . For an  $\omega$ WPDA  $M$ , we introduce the following function  $\|M\|: \Sigma^\omega \rightarrow D$  which is called the *behavior* of  $M$  and which is defined by  $\|M\|(w) = \sum (\text{Val}^\omega(\text{wt}(\rho)) \mid \rho \text{ successful run of } M \text{ on } w)$ .

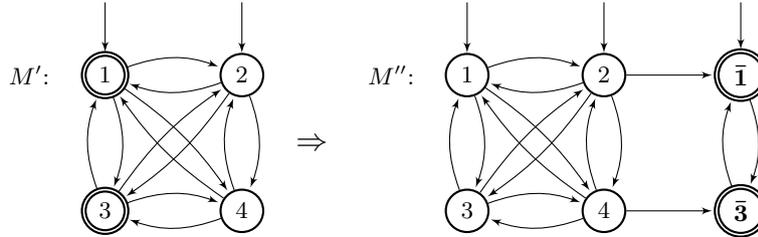
An  $\omega$ PDA or  $\omega$ WPDA  $M$  over  $\Sigma$  is called *unambiguous* if there exists at most one successful run of  $M$  on every word  $w \in \Sigma^\omega$ . If there exists an unambiguous  $\omega$ PDA  $M$  with  $\mathcal{L}(M) = L$ , the language  $L$  is called *unambiguous*.

Any function  $s: \Sigma^\omega \rightarrow D$  is called a *series* over  $\Sigma$  and  $D$ . The set of all such series is denoted by  $D\langle\Sigma^\omega\rangle$ . Every series  $s: \Sigma^\omega \rightarrow D$  which is the behavior of some  $\omega$ WPDA over  $D$  is called  $\omega$ WPDA-*recognizable*.

An  $\omega$ WPDA  $M = (Q, \Gamma, T, I, F, \text{wt})$  that only uses internal transitions, i.e., for which  $\Gamma = \emptyset$  and for all transitions  $t = (q, a, q', s) \in T$  holds  $s = \#$ , is called a weighted finite automaton, or short  $\omega$ WFA. Series that are the behavior of some  $\omega$ WFA are called  $\omega$ WFA-*recognizable*.



■ **Figure 1** Example 5: Weighted  $\omega$ -pushdown automaton over the alphabet  $\Sigma = \{\text{req, ans, call, ret, wait}\}$  and the  $\omega$ -valuation monoid  $\bar{\mathbb{R}}$ . The value after the “:” are the used weights 0 and 1.



■ **Figure 2** Proof of Theorem 6: The states 1, 2, 3, and 4 stand for the set of states that are initial and final, initial but not final, final but not initial, or neither initial nor final, respectively. Groups 1 and 3 are copied into  $\bar{1}$  and  $\bar{3}$ . Transitions into  $\bar{1}$  and  $\bar{3}$  are only allowed from originally non-accepting states.

► **Example 5** ( $\omega$ WPDA). We extend the  $\omega$ -pv-monoid 1 of Example 1 as  $(\bar{\mathbb{R}}, \text{sup}, \text{specialavg}, +, -\infty, 0)$  where we define a new  $\omega$ -valuation function to count and take the average of the counted values. Let  $h$  be a function that maps natural numbers to strings as follows.

$$h: \mathbb{N} \rightarrow \{0, 1\}^*, \quad n \mapsto 0 \underbrace{11 \dots 1}_n 0$$

Then we extend  $h$  to infinite sequences of natural numbers  $h: \mathbb{N}^\omega \rightarrow \{0, 1\}^\omega$  in the natural way. We will consider its inverse where we have for instance  $h^{-1}(011100110011110\dots) = 324\dots$ . Then let  $\text{specialavg} = \lim \text{avg} \circ h^{-1}$ . For  $w \notin (01^*0)^\omega$  we set  $\text{specialavg}(w) = -\infty$ .

Now, we define an automaton  $\mathcal{A}$  as shown in Figure 1. We let  $\mathcal{A} = (\{1, 2\}, \{X, Y\}, T, \{1\}, \{1\}, \text{wt})$  be an  $\omega$ WPDA over the alphabet  $\Sigma = \{\text{req, ans, call, ret, wait}\}$ , where  $T$  is defined as shown in the Figure and the weights are indicated after the colon symbol.

The automaton simulates some kind of (web) server that takes *requests* from clients and *answers* them. For every request, the server has to *call* some amount of other services and *await* their *returns*. Only when all calls have been returned, the server answers the original request. This is a context-free property. Only runs that always eventually return to state 1 to serve new clients are considered valid.

Every call, return, or wait takes one second to operate and this operation time is accounted for in the weight. The specialavg operation sums up all the waiting time per request and returns the long run average response time.

We now state our first main result.

► **Theorem 6.** *Let  $s: \Sigma^\omega \rightarrow D$  be a series. The following are equivalent:*

- $s$  is recognizable by a Büchi-accepting  $\omega$ WPDA,
- $s$  is recognizable by a Muller-accepting  $\omega$ WPDA.

**Proof Idea.** In the direction Büchi to Muller acceptance, the standard approach works also in the weighted case.

For the other direction, the standard approach usually employs a special set of accepting states that have to be traversed to be accepted. This construction needs to be adjusted as it creates infinitely many possible runs in the Büchi automaton for every run of the Muller automaton.

A solution to this problem was presented in [25] whose construction allows exactly one entry point into the special set of accepting states. Entering the group of accepting states is forbidden from a state that is already accepting. In this way, the only successful runs are the ones that switch from the original states to the new group of accepting states at the last possible moment. In contrast to [25], we cannot assume an initially normalized automaton to solve the remaining question of the initial states that are also final.

Instead, the automaton decides non-deterministically if it will eventually see a non-final state in the run. If not, and only in this case, it already starts in the new group of accepting states. Figure 2 depicts the idea of the construction. ◀

### 3 Closure Properties

Let  $\Sigma, \Delta$  be alphabets and  $h: \Sigma \rightarrow \Delta$  a mapping. We can extend  $h$  to infinite words in the natural way by setting  $h(w) = h(a_0)h(a_1)h(a_2) \cdots \in \Delta^\omega$  for  $w = a_0a_1a_2 \cdots \in \Sigma^\omega$ .

Let now  $h: \Delta \rightarrow \Sigma$  and let  $h^{-1}(w) = \{v \in \Delta^\omega \mid h(v) = w\}$ . Then for a series  $s: \Delta^\omega \rightarrow D$ , we define the series  $h(s): \Sigma^\omega \rightarrow D$  by  $h(s)(w) = \sum_{v \in h^{-1}(w)} s(v)$  for all  $w \in \Sigma^\omega$ .

► **Lemma 7.** *Let  $\Sigma, \Delta$  be alphabets,  $(D, +, \text{Val}^\omega, 0)$  an  $\omega$ -valuation monoid and  $h: \Delta \rightarrow \Sigma$  a mapping. If  $s: \Delta^\omega \rightarrow D$  is  $\omega$ WPDA-recognizable, then so is  $h(s): \Sigma^\omega \rightarrow D$ .*

Let  $g: \Sigma \rightarrow D$  be a mapping. We denote by  $\text{Val}^\omega \circ g: \Sigma^\omega \rightarrow D$  the series defined for all  $w \in \Sigma^\omega$  by  $(\text{Val}^\omega \circ g)(w) = \text{Val}^\omega(g(w))$ .

► **Lemma 8.** *Let  $\Sigma$  be an alphabet,  $(D, +, \text{Val}^\omega, 0)$  an  $\omega$ -valuation monoid and  $g: \Sigma \rightarrow D$  a mapping. Then  $\text{Val}^\omega \circ g$  is  $\omega$ WFA-recognizable by an  $\omega$ WFA with only one state.*

Let  $(D, +, \text{Val}^\omega, 0)$  be an  $\omega$ -valuation monoid,  $s: \Sigma^\omega \rightarrow D$  an  $\omega$ WFA-recognizable series and  $L \subseteq \Sigma^\omega$  an  $\omega$ PDA-recognizable language. By  $s \cap L: \Sigma^\omega \rightarrow D$ , we denote the series that assigns the weights of  $s$  to the words accepted by  $L$ . Formally, for words  $u \in \Sigma^\omega$ ,

$$(s \cap L)(u) = \begin{cases} s(u), & \text{if } u \in L \\ 0, & \text{otherwise.} \end{cases}$$

► **Lemma 9.** *Let  $(D, +, \text{Val}^\omega, 0)$  be an  $\omega$ -valuation monoid,  $s: \Sigma^\omega \rightarrow D$  an  $\omega$ WFA-recognizable series and  $L \subseteq \Sigma^\omega$  an  $\omega$ PDA-recognizable language.*

1. *If  $L$  is unambiguous, then the series  $s \cap L: \Sigma^\omega \rightarrow D$  is  $\omega$ WPDA-recognizable.*
2. *If  $D$  is idempotent, then the series  $s \cap L: \Sigma^\omega \rightarrow D$  is  $\omega$ WPDA-recognizable.*

**Proof Idea.** To allow final states of both original Büchi-accepting automata to be visited alternately, we use the standard construction for intersecting unweighted Büchi automata for infinite words, see [32] for details. The assumptions of (1), respectively (2), imply that the weights for  $s \cap L$  are computed correctly (cf. [2]). ◀

Intersection with inherently ambiguous languages over non-idempotent  $\omega$ -valuation monoids might not be  $\omega$ WPDA-recognizable. For a counterexample, consider the  $\omega$ -valuation monoid  $(\mathbb{N}^\infty, +, \prod, 0)$  of natural numbers or  $\infty$  with the natural operations, an inherently ambiguous language like e.g.  $L = \{a^i b^j c^k d^\omega \mid i = j \text{ or } j = k\}$  and intersect it with the

constant series  $s(u) = 1$  for all  $u \in \Sigma^\omega$ . But then, the intersection  $(s \cap L)$  is no longer  $\omega$ WPDA-recognizable which can be seen as follows. An automaton  $M$  for the series  $(s \cap L)$  would yield the value 1 precisely for the words in  $L$ . Since the  $\omega$ -valuation monoid  $\mathbb{N}^\infty$  only allows non-negative integers or  $\infty$  as weights, each word in  $L$  can have only one successful run in  $M$ . Stripping  $M$  of its weights while only keeping transitions with non-zero weight, we obtain an unweighted pushdown automaton  $M'$ . The new automaton  $M'$  has only one successful run for every input  $u \in L$  and is thus unambiguous; moreover,  $M'$  accepts the language  $L$ . This contradicts  $L$  being inherently ambiguous.

► **Definition 10.** Let  $\Sigma$  be an alphabet and  $(D, +, \text{Val}^\omega, 0)$  an  $\omega$ -valuation monoid.

We denote by  $D^{\text{rec}}\langle\langle\Sigma^\omega\rangle\rangle$  the family of  $\omega$ WPDA-recognizable series over  $\Sigma$  and  $D$ . Let further  $D^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$  (with  $\mathcal{N}$  meaning Nivat) denote the set of series  $s$  over  $\Sigma$  and  $D$  such that there exist an alphabet  $\Delta$ , mappings  $h: \Delta \rightarrow \Sigma$  and  $r: \Delta \rightarrow D$  and an  $\omega$ PDA-recognizable language  $L \subseteq \Delta^\omega$  such that

$$s = h((\text{Val}^\omega \circ r) \cap L).$$

We further define  $D_{\text{UNAMB}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$  ( $D_{\text{DET}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$ ) like  $D^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$  with the difference that  $L$  is an unambiguous (deterministic, respectively)  $\omega$ PDA-recognizable language.

► **Example 11.** We extend the  $\omega$ -pv-monoid 1 of Example 1 as  $(\dot{\mathbb{R}}, \text{sup}, \text{partialavg}, +, -\infty, 0)$  where we add a new value  $d$  that will later be ignored, i.e.,  $\dot{\mathbb{R}} = \bar{\mathbb{R}} \cup \{d\}$ . We set  $\text{sup}(-\infty, d) = d$  and  $\text{sup}(r, d) = r$  for every  $r \in \mathbb{R}$ . We define a new  $\omega$ -valuation function to ignore  $d$  and take the average of the remaining values. Let now  $h$  be defined as follows.

$$\begin{aligned} h: \dot{\mathbb{R}} &\rightarrow \bar{\mathbb{R}}^*, & r &\mapsto r, & \text{for } r \in \bar{\mathbb{R}} \\ & & d &\mapsto \epsilon \end{aligned}$$

Then we extend  $h$  to infinite sequences  $h: \dot{\mathbb{R}}^\omega \rightarrow \bar{\mathbb{R}}^\omega$  in the natural way. Now let  $\text{partialavg} = \lim \text{avg} \circ h$  and  $\Sigma = \{a, b\}$ . We make the following definitions:

- $\Delta = \Sigma \times \{0, 1, \dots, 6\}$ ,
- $L = \{(\sigma_1, d_1)(\sigma_2, d_2)(\sigma_3, d_3) \cdots \mid d_i = i \bmod 7, \sigma_i \in \Sigma\}$ ,
- $r(b, i) = d$  for all  $i \in \{0, \dots, 6\}$  and  $r(a, i) = \begin{cases} 1, & \text{if } 5 \leq i \leq 6 \\ 0, & \text{otherwise,} \end{cases}$
- $h(\sigma, i) = \sigma$

The language  $L \subseteq \Delta^\omega$  is obviously  $\omega$ PDA-recognizable. As we will see in the following theorem, the series  $s = h((\text{Val}^\omega \circ r) \cap L) \in \dot{\mathbb{R}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$  is  $\omega$ WPDA-recognizable because  $\dot{\mathbb{R}}$  is idempotent. The series  $s$  calculates the greatest accumulation point of the ratio of events  $a$  happening at the weekend (days 5 and 6) compared to all occurrences of events  $a$ .

The following is the second main Nivat-like decomposition result.

► **Theorem 12.** Let  $\Sigma$  be an alphabet and  $(D, +, \text{Val}^\omega, 0)$  an  $\omega$ -valuation monoid. Then,

$$D^{\text{rec}}\langle\langle\Sigma^\omega\rangle\rangle = D_{\text{DET}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle = D_{\text{UNAMB}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle \subseteq D^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle.$$

If  $D$  is idempotent,  $D^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle = D^{\text{rec}}\langle\langle\Sigma^\omega\rangle\rangle$ .

**Proof.** First, we show  $D^{\text{rec}}\langle\langle\Sigma^\omega\rangle\rangle \subseteq D_{\text{DET}}^{\mathcal{N}}\langle\langle\Sigma^\omega\rangle\rangle$ : Let  $s \in D^{\text{rec}}\langle\langle\Sigma^\omega\rangle\rangle$ . Thus there exists an  $\omega$ WPDA  $M = (Q, \Gamma, T, I, F, wt)$  over  $\Sigma$  such that  $\|M\| = s$ . We will show that there exist  $\Delta$ ,  $h$ ,  $r$  and  $L$  such that  $s = h((\text{Val}^\omega \circ r) \cap L)$ .

Let  $\Delta = T$  and let  $r = \text{wt}: \Delta \rightarrow D$ . We define  $h: \Delta \rightarrow \Sigma$  by  $h((q, a, q', s)) = a$ . Note that the automaton does not allow  $\epsilon$ -transitions and therefore,  $h$  is well-defined. We construct an unweighted  $\omega$ PDA  $M' = (Q, \Gamma, T', I, F)$  over  $\Delta$  with

$$T' = \{(q, (q, a, p, s), p, s) \mid (q, a, p, s) \in T\}.$$

Note that  $M'$  accepts exactly the successful runs of  $M$ . As there is at most one transition of  $M'$  with label  $(q, a, p, s)$ ,  $M'$  is deterministic (and unambiguous). Define  $L = \mathcal{L}(M')$ .

Let  $w \in \Sigma^\omega$ . Therefore,

$$\begin{aligned} h((\text{Val}^\omega \circ r) \cap L)(w) &= \sum (((\text{Val}^\omega \circ r) \cap L)(w') \mid w' \in \Sigma^\omega \text{ and } h(w') = w) \\ &= \sum ((\text{Val}^\omega \circ r)(w') \mid w' \in L \text{ and } h(w') = w) \\ &= \sum (\text{Val}^\omega(\text{wt}(w')) \mid w' \text{ successful run of } M \text{ on } w) \\ &= \|M\|(w) = s(w). \end{aligned}$$

The inclusions  $D_{\text{DET}}^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle \subseteq D_{\text{UNAMB}}^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle \subseteq D^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle$  is true by definition. The converse  $D_{\text{UNAMB}}^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle \subseteq D^{\text{rec}} \langle \langle \Sigma^\omega \rangle \rangle$  is proven by the closure properties of Lemmas 7, 8 and 9(1).

If  $D$  is idempotent, by Lemmas 7, 8 and 9(2), we get  $D^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle \subseteq D^{\text{rec}} \langle \langle \Sigma^\omega \rangle \rangle$ .  $\blacktriangleleft$

The inclusion  $D^{\mathcal{N}} \langle \langle \Sigma^\omega \rangle \rangle \subseteq D^{\text{rec}} \langle \langle \Sigma^\omega \rangle \rangle$  does not hold in general for non-idempotent  $D$ . For the proof, one can consider an adaptation of the counterexample after Lemma 9.

#### 4 Logic for Weighted $\omega$ -Pushdown Automata

The third main goal of this paper is a logical characterization of weighted  $\omega$ -context-free languages. This section introduces this logic. It is based on [15, 17, 29].

Our logic has three components. The first component is a monadic second-order logic (MSO). By Büchi, Elgot, Trakhtenbrot [5, 6, 26, 33], MSO has the same expressive power on finite and infinite words as finite automata.

The second component adds the weights to the logic. Here, this is done by a new layer of formulas that are to be interpreted quantitatively, using the operations of the  $\omega$ -pv-monoid. Formulas of the unweighted part of the logic will be interpreted as  $\mathbb{0}$  or  $\mathbb{1}$  in the  $\omega$ -pv-monoid.

The third component is a dyadic second-order predicate – a binary relation that is called matching relation. Every formula will be allowed to use exactly one such predicate to link positions in words. A matching relation has a specific shape that makes it possible to argue about the stack in pushdown automata or the brackets in Dyck languages or even about the nesting in nested words.

Let  $w \in \Sigma^\omega$ . The set of all positions of  $w$  is  $\mathbb{N}$ . A binary relation  $M \subseteq \mathbb{N} \times \mathbb{N}$  is a *matching* (cf. [29]) if  $M$  is compatible with  $<$ , i.e.,  $(i, j) \in M$  implies  $i < j$ , if each element  $i$  belongs to at most one pair in  $M$ , and if  $M$  is noncrossing, i.e.,  $(i, j) \in M$  and  $(k, l) \in M$  with  $i < k < j$  imply  $i < l < j$ . Let  $\text{Match}(\mathbb{N})$  denote the set of all matchings in  $\mathbb{N} \times \mathbb{N}$ .

Let  $V_1, V_2$  denote countable and pairwise disjoint sets of first-order and second-order variables, respectively. We fix a *matching variable*  $\mu \notin V_1 \cup V_2$ . Let  $\mathcal{V} = V_1 \cup V_2 \cup \{\mu\}$ . Furthermore,  $D$  is always an  $\omega$ -pv-monoid  $(D, +, \text{Val}^\omega, \diamond, \mathbb{0}, \mathbb{1})$ .

► **Definition 13.** Let  $\Sigma$  be an alphabet. The set  $\omega\text{MSO}(D, \Sigma)$  of weighted matching  $\omega$ -MSO formulas over  $\Sigma$  and  $D$  is defined by the extended Backus-Naur form

$$\begin{aligned} \beta ::= & P_a(x) \mid x \leq y \mid x \in X \mid \mu(x, y) \mid \neg\beta \mid \beta \vee \beta \mid \exists x. \beta \mid \exists X. \beta \\ \varphi ::= & d \mid \beta \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus_x \varphi \mid \bigoplus_X \varphi \mid \text{Val}_x \varphi \end{aligned}$$

where  $a \in \Sigma$ ,  $d \in D$ ,  $x, y \in V_1$  and  $X \in V_2$ . We call all formulas  $\beta$  boolean formulas.

■ **Table 1** The semantics of boolean (left) and weighted (right)  $\omega\text{MSO}(D, \Sigma)$  formulas.

$(w, \sigma) \models P_a(x)$ iff $a_{\sigma(x)} = a$	$\llbracket \varphi \oplus \psi \rrbracket(w, \sigma) = \llbracket \varphi \rrbracket(w, \sigma) + \llbracket \psi \rrbracket(w, \sigma)$
$(w, \sigma) \models x \leq y$ iff $\sigma(x) \leq \sigma(y)$	$\llbracket \varphi \otimes \psi \rrbracket(w, \sigma) = \llbracket \varphi \rrbracket(w, \sigma) \diamond \llbracket \psi \rrbracket(w, \sigma)$
$(w, \sigma) \models x \in X$ iff $\sigma(x) \in \sigma(X)$	$\llbracket \bigoplus_x \varphi \rrbracket(w, \sigma) = \sum_{i \in \mathbb{N}} (\llbracket \varphi \rrbracket(w, \sigma[x/i]))$
$(w, \sigma) \models \mu(x, y)$ iff $(\sigma(x), \sigma(y)) \in \sigma(\mu)$	$\llbracket \bigoplus_X \varphi \rrbracket(w, \sigma) = \sum_{I \subseteq \mathbb{N}} (\llbracket \varphi \rrbracket(w, \sigma[X/I]))$
$(w, \sigma) \models \neg \varphi$ iff $(w, \sigma) \not\models \varphi$	$\llbracket \text{Val}_x \varphi \rrbracket(w, \sigma) = \text{Val}^\omega((\llbracket \varphi \rrbracket(w, \sigma[x/i]))_{i \in \mathbb{N}})$
$(w, \sigma) \models \varphi \vee \psi$ iff $(w, \sigma) \models \varphi$ or $(w, \sigma) \models \psi$	$\llbracket \beta \rrbracket(w, \sigma) = \begin{cases} \mathbf{1}, & \text{if } (w, \sigma) \models \beta, \\ \mathbf{0}, & \text{otherwise} \end{cases}$
$(w, \sigma) \models \exists x. \varphi$ iff $\exists j \in \mathbb{N}. (w, \sigma[x/j]) \models \varphi$	
$(w, \sigma) \models \exists X. \varphi$ iff $\exists J \subseteq \mathbb{N}. (w, \sigma[X/J]) \models \varphi$	$\llbracket d \rrbracket(w, \sigma) = d$

Variables denote positions in the word.  $P_a(x)$  is a predicate indicating that the  $x$ -th letter of the word is  $a$ . Furthermore,  $\mu(x, y)$  says that  $x$  and  $y$  will be matched. The operations  $\oplus$  and  $\otimes$  evaluate to the operations  $+$  and  $\diamond$  of the  $\omega$ -pv-monoid  $D$ , respectively. The formulas  $\bigoplus_x$  and  $\bigoplus_X$  sum up over all possible instances of  $x$  and  $X$ , respectively.  $\text{Val}_x \varphi$  applies  $\text{Val}^\omega$  to the sequence of infinitely many  $\varphi$ , each of them instantiated with a position  $x \in \mathbb{N}$ .

Let  $\bar{\mathcal{V}}$  be the collection of all  $\mathcal{V}$ -assignments, i.e., mappings  $\sigma: \mathcal{V} \rightarrow \mathbb{N} \cup 2^{\mathbb{N}} \cup \text{Match}(\mathbb{N})$  where  $\sigma(V_1) \subseteq \mathbb{N}$ ,  $\sigma(V_2) \subseteq 2^{\mathbb{N}}$  and  $\sigma(\mu) \in \text{Match}(\mathbb{N})$ . Let  $\sigma$  be a  $\mathcal{V}$ -assignment. For  $x \in V_1$  and  $j \in \mathbb{N}$ , the update  $\sigma[x/j]$  is the  $\mathcal{V}$ -assignment  $\sigma'$  with  $\sigma'(x) = j$  and  $\sigma'(y) = \sigma(y)$  for all  $y \in \mathcal{V} \setminus \{x\}$ . The updates  $\sigma[X/J]$  and  $\sigma[\mu/M]$  are defined similarly.

Let  $\varphi \in \omega\text{MSO}(D, \Sigma)$  be a formula,  $w = a_0 a_1 a_2 \dots \in \Sigma^\omega$  and let  $\sigma$  be a  $\mathcal{V}$ -assignment. We inductively define  $(w, \sigma) \models \varphi$  if  $\varphi$  is boolean and  $\llbracket \varphi \rrbracket: \Sigma^\omega \times \bar{\mathcal{V}} \rightarrow D$  if  $\varphi$  is non-boolean over the structure of  $\varphi$  as shown in the Table 1, where  $a \in \Sigma$ ,  $d \in D$ ,  $x, y \in V_1$  and  $X \in V_2$ . The logical counterparts  $\wedge$ ,  $\rightarrow$ ,  $\forall x. \varphi$ ,  $\forall X. \varphi$ ,  $x \neq y$ ,  $x < y$  and  $i < j < k$  can be gained from negation and the existing operators.

Note how formulas  $\phi \otimes \psi$  are evaluated by the product operation  $\diamond$  in the  $\omega$ -pv-monoid and also note that our  $\omega\text{WPDAs}$  do not have direct access to this operation. However, the first two layers of our logic, the  $\omega\text{MSO}(D, \Sigma)$  formulas, will be translated into weighted nested  $\omega$ -word automata and simple series of those automata are closed under intersection and therefore,  $\diamond$  can be translated by a product construction.

We now define  $\text{MATCHING}(\mu) \in \omega\text{MSO}(D, \Sigma)$  which ensures that  $\mu \in \text{Match}(\mathbb{N})$ . Let

$$\begin{aligned} \text{MATCHING}(\mu) = & \forall x \forall y. (\mu(x, y) \rightarrow x < y) \wedge \\ & \forall x \forall y \forall k. ((\mu(x, y) \wedge k \neq x \wedge k \neq y) \rightarrow \neg \mu(x, k) \wedge \neg \mu(k, x) \wedge \neg \mu(y, k) \wedge \neg \mu(k, y)) \wedge \\ & \forall x \forall y \forall k \forall l. ((\mu(x, y) \wedge \mu(k, l) \wedge x < k < y) \rightarrow x < l < y). \end{aligned}$$

► **Definition 14.** *The set of formulas of weighted matching  $\omega$ -logic over  $\Sigma$  and  $D$ ,  $\omega\text{ML}(D, \Sigma)$ , denotes the set of all formulas  $\psi$  of the form*

$$\psi = \bigoplus_\mu (\varphi \otimes \text{MATCHING}(\mu)),$$

for short  $\psi = \bigoplus_\mu^{\text{match}} \varphi$ , where  $\varphi \in \omega\text{MSO}(D, \Sigma)$ .

Let  $\psi = \bigoplus_\mu^{\text{match}} \varphi$ ,  $w \in \Sigma^\omega$  and let  $\sigma$  be a  $\mathcal{V}$ -assignment. Then,

$$\llbracket \psi \rrbracket(w, \sigma) = \sum_{M \in \text{Match}(\mathbb{N})} (\llbracket \varphi \rrbracket(w, \sigma[\mu/M])).$$

## 44:10 Nivat-Theorem and Logic for Weighted $\omega$ -Pushdown Automata

Let  $\psi \in \omega\text{ML}(D, \Sigma)$ . We denote by  $\text{Free}(\psi) \subseteq \mathcal{V}$  the set of *free variables* of  $\psi$ . A formula  $\psi$  with  $\text{Free}(\psi) = \emptyset$  is called a *sentence*. For a sentence  $\psi$ ,  $\llbracket \psi \rrbracket(w, \sigma)$  does not depend on  $\sigma$ . It will therefore be omitted and we only write  $\llbracket \psi \rrbracket(w)$  where the series  $\llbracket \psi \rrbracket : \Sigma^\omega \rightarrow D$  is called *defined* by  $\psi$ . A series  $s : \Sigma^\omega \rightarrow D$  is *weighted- $\omega$ ML-definable* if there exists a sentence  $\psi \in \omega\text{ML}(D, \Sigma)$  such that  $\llbracket \psi \rrbracket = s$ .

► **Example 15.** Here we define a logical sentence that defines the same series as in Example 5. Consider the same  $\omega$ -pv-monoid  $(\mathbb{R}, \text{sup}, \text{specialavg}, +, -\infty, 0)$  as there.

The subformula  $p_{\text{structure}}$  ensures that the first symbol is a request and that requests occur directly after answers. The formula  $p_{\text{matching}}$  relates corresponding call and returns and forbids calls without returns and vice versa. Furthermore, calls must be returned before giving the answer to the clients. Finally, the server has to serve clients infinitely often.

$$\begin{aligned} \text{next}(x, y) &= x < y \wedge \neg(\exists z. x \leq z \leq y) \\ \text{first}(x) &= \forall y. x \leq y \\ p_{\text{structure}} &= \forall x. (\text{first}(x) \rightarrow P_{\text{req}}(x)) \wedge \forall x \forall y. \text{next}(x, y) \rightarrow (P_{\text{ans}}(x) \leftrightarrow P_{\text{req}}(y)) \\ p_{\text{matching}} &= \forall x. P_{\text{call}}(x) \rightarrow \exists y. P_{\text{ret}}(y) \wedge \mu(x, y) \\ &\quad \wedge \forall y. P_{\text{ret}}(y) \rightarrow \exists x. P_{\text{call}}(x) \wedge \mu(x, y) \\ &\quad \wedge \forall x. \forall y. [\mu(x, y) \rightarrow \neg(\exists z. x \leq z \leq y \wedge P_{\text{ans}}(z))] \\ &\quad \wedge \forall x. \forall y. [\mu(x, y) \rightarrow ((P_{\text{req}}(x) \wedge P_{\text{ans}}(y)) \vee (P_{\text{call}}(x) \wedge P_{\text{ret}}(y)))] \\ p_{\text{inf\_serving}} &= \forall x. \exists y. (x < y \wedge P_{\text{req}}(y)) \\ \varphi_{\text{unweighted}} &= p_{\text{structure}} \wedge p_{\text{matching}} \wedge p_{\text{inf\_serving}} \end{aligned}$$

The weights of the words are distributed depending on the symbol and the formula  $\varphi_{\text{weighted}}$  also applies the  $\text{Val}^\omega$  function to the resulting sequence of weights.

$$\varphi_{\text{weighted}} = \text{Val}_x [(P_{\text{req}}(x) \vee P_{\text{ans}}(x)) \oplus ((P_{\text{call}}(x) \vee P_{\text{ret}}(x) \vee P_{\text{wait}}(x)) \otimes 1)]$$

Then, we we quantify over the matching variable and only consider the weight calculated in  $\varphi_{\text{weighted}}$  if the formula  $\varphi_{\text{unweighted}}$  is true:

$$\psi = \bigoplus_{\mu}^{\text{match}} \varphi_{\text{unweighted}} \otimes \varphi_{\text{weighted}}$$

Finally, we have  $\llbracket \psi \rrbracket = \|\mathcal{A}\|$  for the  $\omega$ WPDA  $\mathcal{A}$  of Example 5.

The weighted matching  $\omega$ -logic,  $\omega\text{ML}(D, \Sigma)$ , contains exactly one predicate  $\mu$  and exactly one quantification over it. This corresponds to the behavior of pushdown automata where exactly one pushdown tape is used. In contrast, the pushdown automaton uses the  $\omega$ -valuation function  $\text{Val}^\omega$  only once per run and never recursively. As formulas  $\text{Val}_x \text{Val}_y \varphi \in \omega\text{MSO}(D, \Sigma)$  are not always translatable into automata, we follow [11, 17, 22] and define some possible restrictions of our logic.

The set of *almost boolean formulas* is the smallest set of all formulas of  $\omega\text{MSO}(D, \Sigma)$  containing all constants  $d \in D$  and all boolean formulas which is closed under  $\oplus$  and  $\otimes$ .

► **Definition 16** ([11, 22]). Let  $\varphi \in \omega\text{MSO}(D, \Sigma)$ . We call  $\varphi$

1. *strongly- $\otimes$ -restricted* if for all subformulas  $\mu \otimes \nu$  of  $\varphi$ :  
either  $\mu$  and  $\nu$  are almost boolean or  $\mu$  is boolean or  $\nu$  is boolean.
2. *Val-restricted* if for all subformulas  $\text{Val}_x \mu$  of  $\varphi$ ,  $\mu$  is almost boolean.
3. *syntactically restricted* if it is both Val-restricted and strongly- $\otimes$ -restricted.

Let now  $\psi = \bigoplus_{\mu}^{\text{match}} \varphi \in \omega\text{ML}(D, \Sigma)$ . For  $X \in \{\text{strongly-}\otimes, \text{Val}, \text{syntactically}\}$ , we also say that  $\psi$  is *X-restricted* if  $\varphi$  is X-restricted.

The following will be the third main result. *Regular*  $\omega$ -pv-monoids will be defined in the next section on page 11 as they depend on nested  $\omega$ -word automata. We will prove the following theorem in Section 6.

► **Theorem 17.** *Let  $D$  be a regular  $\omega$ -pv-monoid and  $s: \Sigma^\omega \rightarrow D$  be a series. The following are equivalent:*

1.  $s$  is  $\omega$ WPDA-recognizable
2. There is a syntactically restricted  $\omega$ ML( $D, \Sigma$ )-sentence  $\varphi$  with  $\llbracket \varphi \rrbracket = s$ .

## 5 Weighted Nested $\omega$ -Word Languages

The  $\omega$ MSO( $D, \Sigma$ ) formulas correspond exactly to weighted nested  $\omega$ -word languages [11] (cf. [1]). In fact, without considering the existential quantification over the matching relation  $\exists^{\text{match}} \mu$ , the matching must explicitly be encoded in the words; the result is a nested word. Because of limited space, we refrain from a detailed definition of weighted nested  $\omega$ -word automata and refer the reader to [11].

A *nested  $\omega$ -word*  $nw$  over  $\Sigma$  is a pair  $(w, \nu) = (a_0 a_1 a_2 \dots, \nu)$  where  $w \in \Sigma^\omega$  is an  $\omega$ -word and  $\nu \in \text{Match}(\mathbb{N})$  is a matching relation over  $\mathbb{N}$ . Let  $NW^\omega(\Sigma)$  denote the set of all nested  $\omega$ -words over  $\Sigma$ . For two positions  $i, j \in \mathbb{N}$  with  $\nu(i, j)$ , we call  $i$  a *call position* and  $j$  a *return position*. If  $i$  is neither call nor return, we call it an *internal position*. A position  $i$  for  $i \in \mathbb{N}$  is called *top-level* if there exist no positions  $j, k \in \mathbb{N}$  with  $j < i < k$  and  $\nu(j, k)$ .

A *weighted stair Muller nested word automaton* ( $\omega$ WNWA) as defined in [11] is a Muller automaton on nested  $\omega$ -words  $(w, \nu)$  that for every return position has access to the state at the corresponding call position. The stair Muller acceptance condition is a Muller acceptance condition used exclusively on top-level position, i.e., only the states occurring infinitely often in the infinite sequence of top-level positions are considered.

Every function  $s: NW^\omega(\Sigma) \rightarrow D$  is called a *nested  $\omega$ -word series* (nw-series). Every nw-series  $s$  which is the behavior of some  $\omega$ WNWA over  $D$  is called  *$\omega$ WNWA-recognizable*.

We will now discuss how  $\omega$ MSO is an equivalent logic to  $\omega$ WNWAs. Note that  $\omega$ MSO( $D, \Sigma$ ) formulas may contain the free variable  $\mu$ . Given a nested word  $nw = (w, \nu)$ , we let  $\sigma(\mu) = \nu$  and make no difference between  $(w, \sigma) \in \Sigma^\omega \times (\{\mu\} \rightarrow \text{Match}(\mathbb{N}))$  and the nested word  $nw$ . We extend the semantics definitions as follows. Let  $\varphi \in \omega$ MSO( $D, \Sigma$ ) and  $\text{Free}(\varphi) \subseteq \{\mu\}$ , then we define  $\llbracket \varphi \rrbracket_{\text{nw}}: NW^\omega(\Sigma) \rightarrow D$  by letting

$$\llbracket \varphi \rrbracket_{\text{nw}}(w, \nu) = \llbracket \varphi \rrbracket(w, \sigma) \quad \text{for } \sigma(\mu) = \nu.$$

Let  $d \in D$  denote the *constant series* with value  $d$ , i.e.,  $d(nw) = d$  for each  $nw \in NW^\omega(\Sigma)$ .

An  $\omega$ -pv-monoid  $D$  is called *regular* if all constant series of  $D$  are  $\omega$ WNWA-recognizable. In other words,  $D$  is regular if for any alphabet  $\Sigma$ , we have: For each  $d \in D$ , there exists an  $\omega$ WNWA  $A_d$  with  $\|A_d\| = d$ .

Note that for this paper, regularity of  $\omega$ -pv-monoids is defined by the means of  $\omega$ WNWAs. In the proof of Theorem 18, this is used in the structural induction as a logical formula  $\varphi = d$ , for a weight  $d$ , can otherwise not necessarily be translated into an automaton.

Sufficient properties for an  $\omega$ -pv-monoid to be regular are shown in [22]. Especially left-multiplicative and left- $\text{Val}^\omega$ -distributive  $\omega$ -pv-monoids are regular, i.e., if we have  $d \diamond \text{Val}^\omega((d_i)_{i \in \mathbb{N}}) = \text{Val}^\omega((d \diamond d_i)_{i \in \mathbb{N}})$  or  $d \diamond \text{Val}^\omega((d_i)_{i \in \mathbb{N}}) = \text{Val}^\omega(d \diamond d_0, (d_i)_{i \geq 1})$  for all  $d \in D$  and  $(d_i)_{i \in \mathbb{N}} \in D^\omega$ , then  $D$  is regular because we can easily construct  $\omega$ WNWAs (and even  $\omega$ WFAs) for every constant series. All  $\omega$ -pv-monoids in Example 1 are regular.

► **Theorem 18** ([11]). *Let  $D$  be a regular  $\omega$ -pv-monoid and  $s: NW^\omega(\Sigma) \rightarrow D$  be a nw-series. The following are equivalent:*

1.  *$s$  is  $\omega$ WNWA-recognizable*
2. *There is a syntactically restricted  $\omega$ M $SO(D, \Sigma)$ -formula  $\varphi$  with  $\text{Free}(\varphi) \subseteq \{\mu\}$  and  $\llbracket \varphi \rrbracket_{nw} = s$ .*

The mapping  $\pi: NW^\omega(\Sigma) \rightarrow \Sigma^\omega$  removes the nesting relation from the nested word, i.e., for  $nw = (w, \nu)$ , we define  $\pi(nw) = w$ . This can be extended to nw-series  $s: NW^\omega(\Sigma) \rightarrow D$  by setting  $\pi(s)(w) = \sum_{nw \in \pi^{-1}(w)} s(nw)$  which equals  $\pi(s)(w) = \sum_{M \in \text{Match}(\mathbb{N})} s(w, \emptyset[\mu/M])$ .

The following is crucial for the rest of the paper.

► **Lemma 19.** *Let  $s: NW^\omega(\Sigma) \rightarrow D$  be an  $\omega$ WNWA-recognizable nw-series. Then the series  $\pi(s): \Sigma^\omega \rightarrow D$  is  $\omega$ WPDA-recognizable.*

For unweighted languages, there is a similar proof in [4, 15]. Here, the proof is more complicated because the acceptance conditions differ. We have to construct a Büchi-accepting pushdown automaton from a stair Muller nested-word automaton.

**Proof.** By Theorem 6, it suffices to construct a Muller-accepting  $\omega$ WPDA from a given  $\omega$ WNWA. We simulate the transitions of the  $\omega$ WNWA by pushing states onto the stack. Additionally, we enrich the states by the information if we are top-level or not. This information is also pushed onto the stack for the reconstruction of the top-level property upon popping. Furthermore, we allow the new Muller-accepting  $\omega$ WPDA to visit arbitrary subsets of states that are not top-level in between the original Muller-accepting states. ◀

## 6 Equivalence of Logic and Automata

This section proves the equivalence of  $\omega$ ML( $D, \Sigma$ ) and weighted simple  $\omega$ -pushdown automata.

► **Lemma 20.** *Let  $D$  be a regular  $\omega$ -pv-monoid and  $s: \Sigma^\omega \rightarrow D$  be an  $\omega$ WPDA-recognizable series. Then  $s$  is  $\omega$ ML-definable by a syntactically restricted  $\omega$ ML( $D, \Sigma$ )-sentence.*

**Proof.** The proof builds a syntactically restricted  $\omega$ ML( $D, \Sigma$ )-sentence  $\theta$  such that  $\llbracket \theta \rrbracket = s$ . The sentence  $\theta$  defines exactly the behavior of an  $\omega$ WPDA. Hereby, we proceed similarly to [15] and [17, 34, 11]. ◀

► **Lemma 21.** *Let  $D$  be a regular  $\omega$ -pv-monoid and let  $\psi$  be a syntactically restricted  $\omega$ ML( $D, \Sigma$ )-sentence. Then  $\llbracket \psi \rrbracket: \Sigma^\omega \rightarrow D$  is  $\omega$ WPDA-recognizable.*

**Proof.** Let  $\psi = \bigoplus_{\mu}^{\text{Match}} \varphi$  for  $\varphi \in \omega$ M $SO(D, \Sigma)$ . Apply Theorem 18 to infer that  $\llbracket \varphi \rrbracket_{nw}$  is  $\omega$ WNWA-recognizable. Now, we use the projection  $\pi: NW^\omega(\Sigma) \rightarrow \Sigma^\omega$  of Section 5 to get  $\pi(\llbracket \varphi \rrbracket_{nw})(w) = \sum_{M \in \text{Match}(\mathbb{N})} (\llbracket \varphi \rrbracket(w, \emptyset[\mu/M])) = \llbracket \psi \rrbracket(w)$ . By Lemma 19,  $\llbracket \psi \rrbracket = \pi(\llbracket \varphi \rrbracket_{nw})$  is  $\omega$ WPDA-recognizable. ◀

**Proof of Theorem 17.** This is immediate by Lemmas 20 and 21. ◀

## 7 Conclusion

We defined  $\omega$ -pv-monoids and  $\omega$ -pushdown automata with weights from  $\omega$ -pv-monoids. We first generalized a fundamental result of unweighted automata theory: Büchi acceptance and Muller acceptance are expressively equivalent; we can show that this remains the case for weighted simple pushdown automata of infinite words.

For the class of languages recognized by our automata, we proved several closure properties and, as our second main result, a Nivat-like decomposition theorem. It states that the weighted languages in our class are induced by an unweighted context-free language and a very simple weighted part; the two components can be intersected and a projection of this intersection gives us the original language.

The third main result is an expressively equivalent logic. This logic has three layers. The first layer basically describes nested  $\omega$ -word-languages. The first two layers together describe weighted nested  $\omega$ -word-languages. The third layer existentially quantifies the matching variable and corresponds to a projection from nested words to context-free languages. In this way, we can apply the Büchi-Elgot-Trakhtenbrot-Theorem for weighted regular nested  $\omega$ -word-languages to obtain our equivalence result.

The present result raises the question how weighted simple  $\omega$ -pushdown automata on  $\omega$ -valuation monoids and therefore also our weighted matching  $\omega$ -logic relate to a corresponding notion of weighted context-free  $\omega$ -grammars; for weighted simple  $\omega$ -pushdown automata over commutative complete star-omega semirings, this was described in [12].

In Theorem 17, it would be desirable to generalize the notion of regular  $\omega$ -pv-monoids to only require  $\omega$ WPDA instead of  $\omega$ WNWA. The classical inductive proof method of Theorem 18 not longer works in this case. However it seems that  $\omega$ -pv-monoids where constant series are  $\omega$ WPDA-recognizable but not  $\omega$ WNWA-recognizable are very artificial.

---

## References

- 1 R. Alur and P. Madhusudan. Visibly pushdown languages. In *ACM Symposium on Theory of Computing (STOC 2004)*, pages 202–211, 2004. doi:10.1145/1007352.1007390.
- 2 P. Babari and M. Droste. A Nivat theorem for weighted picture automata and weighted MSO logics. *J. Comput. Syst. Sci.*, 104:41–57, 2019. doi:10.1016/j.jcss.2017.02.009.
- 3 C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- 4 A. Blass and Y. Gurevich. A note on nested words. *Microsoft Research*, 2006. URL: <https://www.microsoft.com/en-us/research/publication/180-a-note-on-nested-words/>.
- 5 J. R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi:10.1002/malq.19600060105.
- 6 J. R. Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1966. doi:10.1016/S0049-237X(09)70564-6.
- 7 K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *Computer Science Logic (CSL 2008)*, pages 385–400. Springer, 2008. doi:10.1007/978-3-540-87531-4\_28.
- 8 N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 35: Computer Programming and Formal Systems, pages 118–161. Elsevier, 1963. doi:10.1016/S0049-237X(08)72023-8.
- 9 E. M. Clarke, T. A. Henzinger, H. Veith, and R. P. Bloem. *Handbook of Model Checking*. Springer, 2016. doi:10.1007/978-3-319-10575-8.
- 10 R. S. Cohen and A. Y. Gold. Theory of  $\omega$ -languages I: Characterizations of  $\omega$ -context-free languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977. doi:10.1016/S0022-0000(77)80004-4.
- 11 M. Droste and S. Dück. Weighted automata and logics for infinite nested words. *Information and Computation*, 253:448–466, 2017. doi:10.1016/j.ic.2016.06.010.
- 12 M. Droste, S. Dziadek, and W. Kuich. Greibach normal form for  $\omega$ -algebraic systems and weighted simple  $\omega$ -pushdown automata. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *LIPICs*, pages 38:1–38:14, 2019. doi:10.4230/LIPICs.FSTTCS.2019.38.
- 13 M. Droste, S. Dziadek, and W. Kuich. Weighted simple reset pushdown automata. *Theoretical Computer Science*, 777:252–259, 2019. doi:10.1016/j.tcs.2019.01.016.

- 14 M. Droste, S. Dziadek, and W. Kuich. Greibach normal form for  $\omega$ -algebraic systems and weighted simple  $\omega$ -pushdown automata, 2020. Submitted. [arXiv:2007.08866](https://arxiv.org/abs/2007.08866).
- 15 M. Droste, S. Dziadek, and W. Kuich. Logic for  $\omega$ -pushdown automata. *Information and Computation*, 2020. Special issue on "Weighted Automata", Accepted for publication.
- 16 M. Droste, Z. Ésik, and W. Kuich. The triple-pair construction for weighted  $\omega$ -pushdown automata. In *Conference on Automata and Formal Languages (AFL 2017)*, volume 252 of *Electronic Proceedings in Theoretical Computer Science*, pages 101–113, 2017. doi:10.4204/EPTCS.252.12.
- 17 M. Droste and P. Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007. doi:10.1016/j.tcs.2007.02.055.
- 18 M. Droste and P. Gastin. Weighted automata and weighted logics. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 5, pages 175–211. Springer, 2009. doi:10.1007/978-3-642-01492-5\_5.
- 19 M. Droste and W. Kuich. A Kleene theorem for weighted  $\omega$ -pushdown automata. *Acta Cybernetica*, 23:43–59, 2017. doi:10.14232/actacyb.23.1.2017.4.
- 20 M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009. doi:10.1007/978-3-642-01492-5.
- 21 M. Droste and D. Kuske. Weighted automata. In J.-E. Pin, editor, *Handbook of Automata Theory*, chapter 4. European Mathematical Society, to appear.
- 22 M. Droste and I. Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Information and Computation*, 220:44–59, 2012. doi:10.1016/j.ic.2012.10.001.
- 23 M. Droste and V. Perevoshchikov. A logical characterization of timed pushdown languages. In *Computer Science Symposium in Russia (CSR 2015)*, volume 9139 of *LNCS*, pages 189–203. Springer, 2015. doi:10.1007/978-3-319-20297-6\_13.
- 24 M. Droste and V. Perevoshchikov. Logics for weighted timed pushdown automata. In *Fields of Logic and Computation II*, pages 153–173. Springer, 2015. doi:10.1007/978-3-319-23534-9\_9.
- 25 M. Droste and G. Rahonis. Weighted automata and weighted logics on infinite words. In *Developments in Language Theory (DLT 2006)*, volume 54, pages 49–58. Springer, 2006. doi:10.1007/11779148\_6.
- 26 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–51, 1961. doi:10.2307/2270940.
- 27 Z. Ésik and W. Kuich. A semiring-semimodule generalization of  $\omega$ -context-free languages. In *Theory Is Forever*, volume 3113 of *LNCS*, pages 68–80. Springer, 2004. doi:10.1007/978-3-540-27812-2\_7.
- 28 D. Krob. Monoides et semi-anneaux complets. *Semigroup Forum*, 36:323–339, 1987. doi:10.1007/BF02575025.
- 29 C. Lautemann, T. Schwentick, and D. Thérien. Logics for context-free languages. In *Computer Science Logic (CSL 1994)*, volume 933 of *LNCS*, pages 205–216. Springer, 1994. doi:10.1007/BFb0022257.
- 30 K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. doi:10.1007/978-1-4615-3190-6.
- 31 M. Nivat. Transductions des langages de Chomsky. *Annales de l'Institut Fourier*, 18(1):339–455, 1968. doi:10.5802/aif.287.
- 32 W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 4, pages 133–191. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50009-3.
- 33 B. A. Trakhtenbrot. Finite automata and the logic of single-place predicates. *Doklady Akademii Nauk*, 140(2):326–329, 1961. In Russian. URL: <http://mi.mathnet.ru/dan25511>.
- 34 H. Vogler, M. Droste, and L. Herrmann. A weighted MSO logic with storage behaviour and its Büchi-Elgot-Trakhtenbrot theorem. In *Language and Automata Theory and Applications (LATA 2016)*, volume 9618 of *LNCS*, pages 127–139. Springer, 2016. doi:10.1007/978-3-319-30000-9\_10.

# Synchronization of Deterministic Visibly Push-Down Automata

Henning Fernau 

Universität Trier, Fachbereich IV, Informatikwissenschaften, Germany  
fernau@uni-trier.de

Petra Wolf 

Universität Trier, Fachbereich IV, Informatikwissenschaften, Germany  
<https://www.wolfp.net/>  
wolfp@informatik.uni-trier.de

---

## Abstract

We generalize the concept of synchronizing words for finite automata, which map all states of the automata to the same state, to deterministic visibly push-down automata. Here, a synchronizing word  $w$  does not only map all states to the same state but also fulfills some conditions on the stack content of each run after reading  $w$ . We consider three types of these stack constraints: after reading  $w$ , the stack (1) is empty in each run, (2) contains the same sequence of stack symbols in each run, or (3) contains an arbitrary sequence which is independent of the other runs. We show that in contrast to general deterministic push-down automata, it is decidable for deterministic visibly push-down automata whether there exists a synchronizing word with each of these stack constraints, more precisely, the problems are in EXPTIME. Under the constraint (1), the problem is even in P. For the sub-classes of deterministic very visibly push-down automata, the problem is in P for all three types of constraints. We further study variants of the synchronization problem where the number of turns in the stack height behavior caused by a synchronizing word is restricted, as well as the problem of synchronizing a variant of a sequential transducer, which shows some visibly behavior, by a word that synchronizes the states and produces the same output on all runs.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness; Theory of computation → Grammars and context-free languages; Theory of computation → Automata extensions; Theory of computation → Transducers

**Keywords and phrases** Synchronizing word, Deterministic visibly push-down automata, Deterministic finite automata, Finite-turn push-down automata, Sequential transducer, Computational complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.45

**Related Version** A full version of the paper is available at ArXiv [17], <https://arxiv.org/abs/2005.01374>.

**Funding** *Petra Wolf*: DFG project FE 560/9-1.

## 1 Introduction

The classical *synchronization problem* asks, given a deterministic finite automaton (DFA), whether there exists a *synchronizing word* that brings all states of the automaton to a single state. While this problem is solvable in polynomial time [12, 34, 43], many variants, such as synchronizing only a subset of states [34], or synchronizing a partial automaton without taking an undefined transition (called carefully synchronizing) [25], are PSPACE-complete. Restricting the length of a potential synchronizing word by a parameter in the input also yields a harder problem, namely the NP-complete short synchronizing word problem [31, 16]. The field of synchronizing automata has been intensively studied over the last years, among others in attempt to verify the famous Černý conjecture claiming that every synchronizable DFA admits a synchronizing word of quadratic length in the number of states [12, 13, 39, 40].



© Henning Fernau and Petra Wolf;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 45; pp. 45:1–45:15



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [37, 41]. More information on synchronization of DFA and the Černý conjecture can be found in [43, 7, 1]. In this work, we want to move away from deterministic finite automata to more general deterministic visibly push-down automata.<sup>1</sup>

The synchronization problem has been generalized in the literature to other automata models including infinite-state systems with infinite branching such as weighted and timed automata [15, 36] or register automata [5]. Here, register automata are infinite state systems where a state consists of a control state and register contents.

Another automaton model, where the state set is enhanced with a possibly infinite memory structure, namely a stack, is the class of *nested word automata* (NWAs were introduced in [3]), where an input word is enhanced with a matching relation determining at which pair of positions in a word a symbol is pushed to and popped from the stack. The class of languages accepted by NWAs is identical to the class of *visibly push-down languages* (VPL) accepted by *visibly push-down automata* (VPDA) and forms a proper sub-class of the deterministic context-free languages. VPDAs have first been studied by Mehlhorn [27] under the name *input-driven pushdown automata* and became quite popular more recently due to the work by Alur and Madhusudan [2], showing that VPLs share several nice properties with regular languages. For more on VPLs we refer to the survey [30]. In [14], the synchronization problem for NWAs was studied. There, the concept of synchronization was generalized to bringing all states to one single state such that for all runs the stack is empty (or in its start configuration) after reading the synchronizing word. In this setting, the synchronization problem is solvable in polynomial time (again indicating similarities of VPLs with regular languages), while the short synchronizing word problem (with length bound given in binary) is PSPACE-complete; the question of synchronizing from or into a subset is EXPTIME-complete. Also, matching exponential upper bounds on the length of a synchronizing word are given.

Our attempt in this work is to study the synchronization problem for real-time (no  $\epsilon$ -transitions) deterministic visibly push-down automata (DVPDA) and several sub-classes thereof, like real-time deterministic very visibly push-down automata (DVVPDA for short; this model was introduced in [24]), real-time deterministic visibly counter automata (DVCA for short; this model appeared a.o. in [6, 38, 9, 21, 22, 23]) and finite turn variants thereof. We want to point out that, despite the equivalence of the accepted language class, the automata models of nested word automata and visibly push-down automata still differ and the results from [14] do not immediately transfer to VPDAs, as for NWAs an input word is equipped with a matching relation, which VPDAs lack of. In general, the complexity of the synchronization problem can differ for different automata models accepting the same language class. For instance, in contrast to the polynomial time solvable synchronization problem for DFAs, the generalized synchronization problem for finite automata with one ambiguous transition is PSPACE-complete, as well as the problem of carefully synchronizing a DFA with one undefined transition [26]. We will not only consider the synchronization model introduced in [14], where reading a synchronizing word results in an empty stack on all runs; but we will also consider a synchronization model where not only the final state on every run must be the same but also the stack content needs to be identical, as well as a model where only the states needs to be synchronized and the stack content might be

---

<sup>1</sup> The term *synchronization of push-down automata* already occurs in the literature, i.e., in [11, 4], but there the term *synchronization* refers to some relation of the input symbols to the stack behavior [11] or to reading different words in parallel [4]; not to confuse it with our notion of synchronizing states.

arbitrary. These three models of synchronization have been introduced in [28], where length bounds on a synchronizing word for general DPDAs have been studied dependent on the stack height. The complexity of these three concepts of synchronization for general DPDAs are considered in [18], where it is shown that synchronizability is undecidable for general DPDAs and deterministic counter automata (DCA). It becomes decidable for deterministic partially blind counter automata and is PSPACE-complete for some types of finite turn DPDAs, while it is still undecidable for other types of finite turn DPDAs.

In contrast, we will show in the following that for DVPDAs and considered sub-classes hereof, the synchronization problem for all three stack models, with restricted or unrestricted number of turns, is in EXPTIME and hence decidable. For DVVPDAs and DVCAs, the synchronization problems for all three stack models (with unbounded number of turns) are even in P. Like the synchronization problem for NWA's in the empty stack model considered in [14], we observe that the synchronization problem for DVPDAs in the empty stack model is solvable in polynomial time, whereas synchronization of DVPDAs in the same and arbitrary stack models is at least PSPACE-hard. If the number of turns caused by a synchronizing word on each run is restricted, the synchronization problem becomes PSPACE-hard for all considered automata models for  $n > 0$  and is only in P for  $n = 0$  in the empty stack model. We will further introduce variants of synchronization problems distinguishing the same and arbitrary stack models by showing complementary complexities in these models. For problems considered in [18], these two stack models have always shared their complexity status.

Due to lack of space, missing proof details can be found in the long version of this work [17].

## 2 Fixing Notations

We refer to the empty word as  $\epsilon$ . For a finite alphabet  $\Sigma$ , we denote with  $\Sigma^*$  the set of all words over  $\Sigma$  and with  $\Sigma^+ = \Sigma\Sigma^*$  the set of all non-empty words. For  $i \in \mathbb{N}$ , we set  $[i] = \{1, 2, \dots, i\}$ . For  $w \in \Sigma^*$ , we denote with  $|w|$  the length of  $w$ , with  $w[i]$  for  $i \in [|w|]$  the  $i$ 'th symbol of  $w$ , and with  $w[i..j]$  for  $i, j \in [|w|]$  the subword  $w[i]w[i+1]\dots w[j]$  of  $w$ . We call  $w[1..i]$  a prefix and  $w[i..|w|]$  a suffix of  $w$ . If  $i < j$ , then  $w[j, i] = \epsilon$ .

We call  $A = (Q, \Sigma, \delta, q_0, F)$  a *deterministic finite automaton* (DFA for short) if  $Q$  is a finite set of states,  $\Sigma$  is a finite input alphabet,  $\delta$  is a transition function  $Q \times \Sigma \rightarrow Q$ ,  $q_0$  is the initial state, and  $F \subseteq Q$  is the set of final states. The transition function  $\delta$  is generalized to words by  $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$  for  $w \in \Sigma^*$ . A word  $w \in \Sigma^*$  is accepted by  $A$  if  $\delta(q_0, w) \in F$  and the language accepted by  $A$  is defined by  $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$ . We extend  $\delta$  to sets of states  $Q' \subseteq Q$  or to sets of letters  $\Sigma' \subseteq \Sigma$ , letting  $\delta(Q', \Sigma') = \{\delta(q', \sigma') \mid (q', \sigma') \in Q' \times \Sigma'\}$ . Similarly, we may write  $\delta(Q', \Sigma') = p$  to define  $\delta(q', \sigma') = p$  for each  $(q', \sigma') \in Q' \times \Sigma'$ . The synchronization problem for DFAs (called DFA-SYNC) asks for a given DFA  $A$  whether there exists a synchronizing word for  $A$ . A word  $w$  is called a *synchronizing word* for a DFA  $A$  if it brings all states of the automaton to one single state, i.e.,  $|\delta(Q, w)| = 1$ .

We call  $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$  a *deterministic push-down automaton* (DPDA for short) if  $Q$  is a finite set of states; the finite sets  $\Sigma$  and  $\Gamma$  are the input and stack alphabet, respectively;  $\delta$  is a transition function  $Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$ ;  $q_0$  is the initial state;  $\perp \in \Gamma$  is the stack bottom symbol which is only allowed as the first (lowest) symbol in the stack, i.e., if  $\delta(q, a, \gamma) = (q', \gamma')$  and  $\gamma'$  contains  $\perp$ , then  $\perp$  only occurs in  $\gamma'$  as its prefix and moreover,  $\gamma = \perp$ ; and  $F$  is the set of final states. We will only consider *real-time* push-down automata and forbid  $\epsilon$ -transitions, as can be seen in the definition. Notice that the bottom symbol can be removed, but then the computation gets stuck.

Following [14], a *configuration* of  $M$  is a tuple  $(q, v) \in Q \times \Gamma^*$ . For a letter  $\sigma \in \Sigma$  and a stack content  $v$ , with  $|v| = n$ , we write  $(q, v) \xrightarrow{\sigma} (q', v[1..(n-1)]\gamma)$  if  $\delta(q, \sigma, v[n]) = (q', \gamma)$ . This means that the top of the stack  $v$  is the right end of  $v$ . We also denote with  $\longrightarrow$  the reflexive transitive closure of the union of  $\xrightarrow{\sigma}$  over all letters in  $\Sigma$ . The input words on top of  $\longrightarrow$  are concatenated accordingly, so that  $\longrightarrow = \bigcup_{w \in \Sigma^*} \xrightarrow{w}$ . The language  $\mathcal{L}(M)$  accepted by a DPDA  $M$  is  $\mathcal{L}(M) = \{w \in \Sigma^* \mid (q_0, \perp) \xrightarrow{w} (q_f, \gamma), q_f \in F\}$ . We call the sequence of configurations  $(q, \perp) \xrightarrow{w} (q', \gamma)$  the *run* induced by  $w$ , starting in  $q$ , and ending in  $q'$ . We might also call  $q'$  the *final state* of the run.

We will discuss three different concepts of synchronizing DPDAs. For all concepts, we require that a synchronizing word  $w \in \Sigma^*$  maps all states, starting with an empty stack, to the same synchronizing state, i.e., for all  $q, q' \in Q$ :  $(q, \perp) \xrightarrow{w} (\bar{q}, v)$ ,  $(q', \perp) \xrightarrow{w} (\bar{q}, v')$ . In other words, for a synchronizing word all runs started on some states in  $Q$  end up in the same state. In addition to synchronizing the states of a DPDA, we will consider the following two conditions for the stack content: (1)  $v = v' = \perp$ , (2)  $v = v'$ . We will call (1) the *empty stack model* and (2) the *same stack model*. In the third case, we do not put any restrictions on the stack content and call this the *arbitrary stack model*.

As we are only interested in synchronizing a DPDA, we can neglect the start and final states.

Starting from DPDAs, we define the following sub-classes thereof:

- A *deterministic visibly push-down automaton* (DVPDA) is a DPDA where the input alphabet  $\Sigma$  can be partitioned into  $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$  such that the change in the stack height is determined by the partition of the alphabet. To be more precise, the transition function  $\delta$  is modified such that it can be partitioned accordingly into  $\delta = \delta_c \cup \delta_i \cup \delta_r$  such that  $\delta_c: Q \times \Sigma \rightarrow Q \times (\Gamma \setminus \{\perp\})$  puts a symbol on the stack,  $\delta_i: Q \times \Sigma \rightarrow Q$  leaves the stack unchanged, and  $\delta_r: Q \times \Sigma \times \Gamma \rightarrow Q$  reads and pops a symbol from the stack [2]. If  $\perp$  is the symbol on top of the stack, then  $\perp$  is only read and not popped. We call letters in  $\Sigma_{\text{call}}$  *call* or *push* letters; letter in  $\Sigma_{\text{int}}$  *internal* letters; and letters in  $\Sigma_{\text{ret}}$  *return* or *pop* letters. The language class accepted by DVPDA is equivalent to the class of languages accepted by deterministic nested word automata (see [14]).
- A *deterministic very visibly push-down automaton* (DVVPA) is a DVPDA where not only the stack height but also the stack content is completely determined by the input alphabet, i.e., for a letter  $\sigma \in \Sigma$  and all states  $p, q \in Q$  for  $\delta_c(p, \sigma) = (p', \gamma_p)$  and  $\delta_c(q, \sigma) = (q', \gamma_q)$  it holds that  $\gamma_p = \gamma_q$ .
- A *deterministic visibly (one) counter automaton* (DVCA) is a DVPDA where  $|\Gamma \setminus \{\perp\}| = 1$ ; note that every DVCA is also a DVVPA.

We are now ready to define a family of synchronization problems, the complexity of which will be our subject of study in the following sections.

► **Definition 1** (SYNC-DVPDA-EMPTY).

*Given:* DPDA  $M = (Q, \Sigma, \Gamma, \delta, \perp)$ .

*Question:* Does there exist a word  $w \in \Sigma^*$  that synchronizes  $M$  in the empty stack model?

For the same stack model, we refer to the synchronization problem above as SYNC-DVPDA-SAME and as SYNC-DVPDA-ARB in the arbitrary stack model. Variants of these problems are defined by replacing the DVPDA in the definition above by a DVVPA, and DVCA. If results hold for several stack models or automata models, then we summarize the problems by using set notations in the corresponding statements. For the problems SYNC-DVPDA-SAME and SYNC-DVPDA-ARB, we introduce two further refined variants of these problems, denoted by the extension -RETURN and -NORETURN, where for all input DVPDA in the former variant  $\Sigma_{\text{ret}} \neq \emptyset$  holds, whereas in the latter variant  $\Sigma_{\text{ret}} = \emptyset$  holds. In the following,

■ **Table 1** Complexity status of the synchronization problem for different classes of deterministic real-time visibly push-down automata in different stack synchronization modes. For the  $n$ -turn synchronization variants,  $n$  takes all values not explicitly listed. All our problems are in EXPTIME.

class of automata	empty stack model	same stack model	arbitrary stack model
DVPDA	P	PSPACE-complete	PSPACE-hard
DVPDA-NoReturn	P	PSPACE-complete	P
DVPDA-Return	P	P	PSPACE-hard
$n$ -Turn-Sync-DVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVPDA	P	PSPACE-complete	PSPACE-complete
DVVPDA	P	P	P
$n$ -Turn-Sync-DVVPDA	PSPACE-hard	PSPACE-hard	PSPACE-hard
0-Turn-Sync-DVVPDA	P	PSPACE-complete	PSPACE-complete
DVCA	P	P	P
$n$ -Turn-Sync-DVCA	PSPACE-hard	PSPACE-hard	PSPACE-hard
1-Turn-Sync-DVCA	PSPACE-complete	PSPACE-complete	PSPACE-complete
0-Turn-Sync-DVCA	P	PSPACE-complete	PSPACE-complete

these variants reveal insights in the differences between synchronization in the same stack and arbitrary stack models, as well as connections to a concept of trace-synchronizing a sequential transducer showing some visibly behavior.

We will further consider synchronization of these automata classes in a finite-turn setting. Finite-turn push-down automata were introduced in [20]. We adopt the definition in [42]. For a DVPDA  $M$ , an *upstroke* of  $M$  is a sequence of configurations induced by an input word  $w$  such that no transition decreases the stack-height. Accordingly, a *downstroke* of  $M$  is a sequence of configurations in which no transition increases the stack-height. A stroke is either an upstroke or downstroke. A DVPDA  $M$  is an  $n$ -turn DVPDA if for all  $w \in \mathcal{L}(M)$  the sequence of configurations induced by  $w$  can be split into at most  $n + 1$  strokes. Especially, for 1-turn DVPDAs, each sequence of configurations induced by an accepting word consists of one upstroke followed by a most one downstroke. Two subtleties arise when translating this concept to synchronization: (a) there is no initial state so that there is no way to associate a stroke counter with a state, and (b) there is no language of accepted words that restricts the set of words on which the number of strokes should be limited. Hence, in the synchronization setting the finite turn property is not a property of the push-down automaton but rather of the word applied to all states in parallel. We therefore generalize the concept of finite-turn DVPDAs to finite-turn synchronization for DVPDAs as follows.

► **Definition 2** ( $n$ -TURN-SYNC-DVPDA-EMPTY).

Given: DVPDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ .

Question: Is there a synchronizing word  $w \in \Sigma^*$  in the empty stack model, such that for all states  $q \in Q$ , the sequence of configurations  $(q, \perp) \xrightarrow{w} (\bar{q}, \perp)$  consists of at most  $n + 1$  strokes?

We call such a synchronizing word  $w$  an  $n$ -turn synchronizing word for  $M$ . We define  $n$ -TURN-SYNC-DVPDA-SAME and  $n$ -TURN-SYNC-DVPDA-ARB accordingly for the same stack and arbitrary stack model. Further, we extend the problem definition to other classes of automata such as real-time DVVPDAs, and DVCA. Table 1 summarizes our results, obtained in the next sections, on the complexity status of these problems together with the above introduced synchronization problems.

Finally, we introduce two PSPACE-complete problems for DFAs to reduce from later.

► **Definition 3** (DFA-SYNC-INTO-SUBSET (PSPACE-complete [32])).

Given: DFA  $A = (Q, \Sigma, \delta)$ , subset  $S \subseteq Q$ .

Question: Is there a word  $w \in \Sigma^*$  such that  $\delta(Q, w) \subseteq S$ ?

► **Definition 4** (DFA-SYNC-FROM-SUBSET (PSPACE-complete [34])).

Given: DFA  $A = (Q, \Sigma, \delta)$  with  $S \subseteq Q$ .

Question: Is there a word  $w \in \Sigma^*$  that synchronizes  $S$ , i.e., for which  $|\delta(S, w)| = 1$  is true?

### 3 DVPDAs – Distinguishing the Stack Models

We start with some positive result showing that we come down from the undecidability of the synchronization problem for general DPDAs in the empty set model to a polynomial time solvable version by considering visibly DPDAs.

► **Theorem 5.** *The problems SYNC-DVPDA-EMPTY, SYNC-DVCA-EMPTY, and SYNC-DVVPDA-EMPTY are decidable in polynomial time.*

**Proof.** We prove the claim for SYNC-DVPDA-EMPTY as the other automata classes are sub-classes of DVPDAs. Let  $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$  be a DVPDA. First, observe that if  $\Sigma_{\text{ret}}$  is empty, then any synchronizing word  $w$  for  $M$  in the empty stack model cannot contain any letter from  $\Sigma_{\text{call}}$ . Hence,  $M$  is basically a DFA and for DFAs the synchronization problem is in P [12, 43, 34]. From now on, assume  $\Sigma_{\text{ret}} \neq \emptyset$ . We show that a pair argument similar to the one for DFAs can be applied, namely that  $M$  is synchronizable in the empty stack model if and only if every pair of states  $p, q \in Q$  can be synchronized in the empty stack model. The only if direction is clear as every synchronizing word for  $Q$  also synchronizes each pair of states. For the other direction, observe that since  $M$  is a DVPDA, the stack height of each path starting in any state of  $M$  is predefined by the sequence of input symbols. Hence, if we focus on the two runs starting in  $p, q$  and ensure that their stacks are empty after reading a word  $w$ , then also the stacks of all other runs starting in other states in parallel are empty after reading  $w$ . Therefore, we can successively concatenate words that synchronize some pair of active states in the empty stack model and end up with a word that synchronizes all states of  $M$  in the empty stack model. Further formal algorithmic details can be found in the long version [17]. ◀

Does this mean everything is easy and we are done? Interestingly, the picture is not that simple, as considering the same and arbitrary stack models shows.

► **Theorem 6.** *The problem SYNC-DVPDA-SAME is PSPACE-hard.*

**Proof.** We reduce from DFA-SYNC-INTO-SUBSET. Let  $A = (Q, \Sigma, \delta)$  be a DFA and  $S \subseteq Q$ . We construct from  $A$  a DVPDA  $M = (Q \cup \{q_S\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{\ominus, \ominus, \perp\}, \delta' = \delta'_c \cup \delta'_i \cup \delta'_r, \perp)$  with  $q_S \notin Q$ ,  $\Sigma_{\text{call}} = \{a\}$ ,  $\Sigma_{\text{int}} = \Sigma$ ,  $\Sigma_{\text{ret}} = \emptyset$  and  $\Sigma_{\text{call}} \cap \Sigma_{\text{int}} = \emptyset$ . The transition function  $\delta'_i$  agrees with  $\delta$  on all letters in  $\Sigma_{\text{int}}$ . For  $q_S$ , we set  $\delta'_c(q_S, a) = (q_S, \ominus)$  and  $\delta'_i(q_S, \sigma) = q_S$  for all  $\sigma \in \Sigma_{\text{int}}$ . For  $q \in S$ , we set  $\delta'_c(q, a) = (q_S, \ominus)$ , and for  $q \notin S$ ,  $\delta'_c(q, a) = (q, \ominus)$ .

Note that  $q_S$  is a sink-state and can only be reached from states in  $S$  with a transition by the call-letter  $a$ . For states not in  $S$ , the input letter  $a$  pushes an  $\ominus$  on the stack which cannot be pushed to the stack by any letter on a path starting in  $q_S$ . Hence, in order to synchronize  $M$  in the same stack model, a letter  $a$  might only and must be read in a configuration where only states in  $S \cup \{q_S\}$  are active. Every word  $w \in \Sigma_{\text{int}}^*$  that brings  $M$  in such a configuration also synchronizes  $Q$  in  $A$  into the set  $S$ . ◀

From the proof of Theorem 6, we can conclude the next results by observing that a DVPDA without any return letter cannot make any turn.

► **Corollary 7.** *SYNC-DVPDA-SAME-NORETURN and 0-TURN-SYNC-DVPDA-SAME are PSPACE-hard.*

In contrast with the two previous results, SYNC-DVPDA-SAME is solvable in polynomial time if we have the promise that  $\Sigma_{\text{ret}} \neq \emptyset$ .

► **Theorem 8.** *SYNC-DVPDA-SAME-RETURN is in P.*

**Proof.** We prove the claim by straight reducing to SYNC-DVPDA-EMPTY with the identity function. If a DVPDA  $M$  with  $\Sigma_{\text{ret}} \neq \emptyset$  can be synchronized in the same stack model with a synchronizing word  $w$ , then  $w$  can be extended to  $ww'$  where  $w' \in \Sigma_{\text{ret}}^*$  empties the stack. As  $M$  is deterministic and complete,  $w'$  is defined on all states. As after reading  $w$ , the stack content on all paths is the same, reading  $w'$  extends all paths with the same sequence of states. Conversely, a word  $w$  synchronizing a DVPDA  $M$  with  $\Sigma_{\text{ret}} \neq \emptyset$  in the empty stack model also synchronizes  $M$  in the same stack model. ◀

The arbitrary stack model requires the most interesting construction in the following proof.

► **Theorem 9.** *SYNC-DVPDA-ARB is PSPACE-hard.*

**Proof.** We give a reduction from the PSPACE-complete problem DFA-SYNC-FROM-SUBSET. Let  $A = (Q, \Sigma, \delta)$  be a DFA with  $S \subseteq Q$ . We construct from  $A$  a DVPDA  $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, Q \cup \{\perp\}, \delta' = \delta'_c \cup \delta'_i \cup \delta'_r, \perp)$  where all unions in the definition of  $M$  are disjoint. Let  $\Sigma_{\text{call}} = \Sigma$ ,  $\Sigma_{\text{int}} = \emptyset$ , and  $\Sigma_{\text{ret}} = \{r\}$  with  $r \notin \Sigma$ .

For states  $s \in S$ , we set  $\delta'_r(s, r, \perp) = s$  and for states  $q \in Q \setminus S$ , we set  $\delta'_r(q, r, \perp) = t$  for some arbitrary but fixed  $t \in S$ . For states  $p, q \in Q$ , we set  $\delta'_r(q, r, p) = p$ .

For each call letter  $\sigma \in \Sigma_{\text{call}}$ , we set for  $q \in Q$ ,  $\delta'_c(q, \sigma) = (\delta(q, \sigma), q)$ .

First, assume  $w$  is a word that synchronizes the set  $S$  in the DFA  $A$ . Then, it can easily be observed that  $rw$  is a synchronizing word for  $M$  in the arbitrary stack model.

Now, assume  $w$  is a synchronizing word for  $M$  in the arbitrary stack model. If  $w \in \Sigma_{\text{call}}^*$ , then  $w$  is also a synchronizing word for  $A$  and especially synchronizes the set  $S$  in  $A$ . (\*) Next, assume  $w$  contains some letters  $r$ . The action of  $r$  is designed such that it maps  $Q$  to the set  $S$  if applied to an empty stack and otherwise gradually undoes the transitions performed by letters from  $\Sigma_{\text{call}}$ . This is possible as each letter  $\sigma \in \Sigma_{\text{call}}$  stores its pre-image on the stack when  $\sigma$  is applied. Further,  $r$  acts as the identity on the states in  $S$  if applied to an empty stack. Hence, whenever the stacks are empty while reading some word, all states in  $S$  are active.

Hence, if  $\sigma r$  is a subword of a synchronizing word  $w = u\sigma r v$  of  $M$ , with  $\sigma \in \Sigma_{\text{call}}$ , then  $w' = uv$  is also a synchronizing word of  $M$ . This justifies the set of rewriting rules  $R = \{\sigma r \rightarrow \varepsilon \mid \sigma \in \Sigma_{\text{call}}\}$ . Now, consider a synchronizing word  $w$  of  $M$  where none of the rewriting rules from  $R$  applies, and, which by (\*) contains some letter  $r$ . Hence,  $w \in \{r\}^* \Sigma_{\text{call}}^*$ . By (\*),  $w = r^k v$ , with  $k > 0$ , and  $v \in \Sigma_{\text{call}}^*$ . Then,  $w' = rv$  is also a synchronizing word of  $M$ , because for all states  $q \in Q$ ,  $M$  is in the same configuration after reading  $r$ , starting in configuration  $(q, \perp)$ , as after reading  $rr$ . But as only (and all) states from  $S$  are active after reading  $r$ ,  $v$  is also a word in  $\Sigma^*$  that synchronizes the set  $S$  in  $A$ . ◀

Observe that in the construction above,  $\Sigma_{\text{ret}} \neq \emptyset$  for all input DFAs. The next corollary follows from Theorem 9 and should be observed together with the next theorem in contrast to Theorem 8 and Corollary 7.

► **Corollary 10.** *SYNC-DVPDA-ARB-RETURN is PSPACE-hard.*

► **Theorem 11.** *SYNC-DVPDA-ARB-NORETURN  $\equiv$  DFA-SYNC.*

**Proof.** Let  $M$  be a DVPDA with empty set of return symbols. As there is no return-symbol, the transitions of  $M$  cannot depend on the stack content. Hence, we can redistribute the symbols in  $\Sigma_{\text{call}}$  into  $\Sigma_{\text{int}}$  and obtain a DFA. The converse is trivial. ◀

If we move from deterministic visibly push-down automata to even more restricted classes, like deterministic very visibly push-down automata or deterministic visibly counter automata, the three stack models do no longer yield synchronization problems with different complexities. Instead, all three models are equivalent, as stated next. Hence, their synchronization problems can be solved by the pair-argument presented in Theorem 5 in polynomial time.

► **Theorem 12.** *SYNC-DVCA-EMPTY  $\equiv$  SYNC-DVCA-SAME  $\equiv$  SYNC-DVCA-ARB.  
SYNC-DVVPDA-EMPTY  $\equiv$  SYNC-DVVPDA-SAME  $\equiv$  SYNC-DVVPDA-ARB.*

**Proof.** First, note that every DVCA is also a DVVPDA. If for a DVVPDA  $\Sigma_{\text{ret}} \neq \emptyset$ , then we can empty the stack after synchronizing the state set, as the very visibly conditions ensures that the contents of the stacks on all runs coincide. As the automaton is deterministic, all transitions for letters in  $\Sigma_{\text{ret}}$  are defined on each state. As the stack content on all runs coincides in every step, the arbitrary stack model is identical to the same stack model and hence equivalent to the empty stack model. If  $\Sigma_{\text{ret}} = \emptyset$ , then we can reassign  $\Sigma_{\text{call}}$  to  $\Sigma_{\text{int}}$  in order to reduce from the same-stack and arbitrary stack to the empty stack variant, as transitions cannot depend on the stack content which is again the same on all runs due to the very visibly condition. ◀

#### 4 Restricting the Number of Turns Makes Synchronization Harder

Let us now restrict the number of turns a synchronizing word may cause on any run. Despite the fact that we are hereby restricting the considered model even further, the synchronization problem becomes even harder, in contrast to the previous section.

► **Theorem 13.** *For every fixed  $n \in \mathbb{N}$  with  $n > 0$ , the problems  $n$ -TURN-SYNC-DVCA-SAME and  $n$ -TURN-SYNC-DVCA-ARB are PSPACE-hard.*

**Proof.** We give a reduction from the PSPACE-complete problem DFA-SYNC-INTO-SUBSET. Let  $A = (Q, \Sigma, \delta)$  be a DFA with  $S \subseteq Q$ . We construct from  $A$  a DVCA  $M = (Q \cup \{q_{\text{sync}}\} \cup \{q_{\text{stall}_i} \mid 0 \leq i \leq n\}, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \{1, \perp\}, \delta' = \delta'_c \cup \delta'_i \cup \delta'_r, \perp)$ , where all unions are disjoint. We set  $\Sigma_{\text{int}} = \Sigma$ ,  $\Sigma_{\text{call}} = \{a\}$  and  $\Sigma_{\text{ret}} = \{b\}$ . For all internal letters,  $\delta'_i$  agrees with  $\delta$  on all states in  $Q$ . For the letter  $a$ , we set for all  $q \in S$ ,  $\delta'_c(q, a) = (q_{\text{stall}_0}, 1)$  and for all  $q \in Q \setminus S$ , we set  $\delta'_c(q, a) = (q, 1)$ . For  $b$ , we loop in every state in  $Q$ . For  $q_{\text{sync}}$ , we loop with every letter in  $q_{\text{sync}}$  (incrementing the counter with  $a$  and decrementing it with  $b$ ).

Let  $r$  be an arbitrary but fixed state in  $Q$ . For the states  $q_{\text{stall}_i}$ , we set for  $i < n$ ,  $\delta'_c(q_{\text{stall}_i}, a) = (q_{\text{stall}_i}, 1)$ . Further, for even index  $i < n$ , we set  $\delta'_r(q_{\text{stall}_i}, b, 1) = q_{\text{stall}_{i+1}}$  and  $\delta'_r(q_{\text{stall}_i}, b, \perp) = r$ . For odd index  $i < n$ , we set  $\delta'_r(q_{\text{stall}_i}, b, 1) = r$ , and  $\delta'_r(q_{\text{stall}_i}, b, \perp) = q_{\text{stall}_{i+1}}$ . For even  $n$ , let  $\delta'_c(q_{\text{stall}_n}, a) = (q_{\text{sync}}, 1)$ ,  $\delta'_r(q_{\text{stall}_n}, b, 1) = r$ , and  $\delta'_r(q_{\text{stall}_n}, b, \perp) = r$ . For odd  $n$ , let  $\delta'_c(q_{\text{stall}_n}, a) = (q_{\text{stall}_n}, 1)$ ,  $\delta'_r(q_{\text{stall}_n}, b, 1) = r$ , and  $\delta'_r(q_{\text{stall}_n}, b, \perp) = q_{\text{sync}}$ . All other transitions (on internal letters) act as the identity.

Observe that the state  $q_{\text{sync}}$  must be the synchronizing state of  $M$ , since it is a sink state. In order to reach  $q_{\text{sync}}$  from any state in  $Q$ , the automaton must pass through all the states  $q_{\text{stall}_i}$  for all  $0 \leq i \leq n$ , by construction. Since we can only pass from a state  $q_{\text{stall}_i}$  to

$q_{\text{stall}_{i+1}}$  with an empty or non-empty stack in alternation, passing through all states  $q_{\text{stall}_i}$ , for  $0 \leq i \leq n$ , forces  $M$  to make  $n$  turns. For even  $n$ , the last upstroke is enforced by passing from  $q_{\text{stall}_n}$  to  $q_{\text{sync}}$  by explicitly increasing the stack. As  $M$  is only allowed to make  $n$  turns while reading the  $n$ -turn synchronizing word it follows that any of the states  $q_{\text{stall}_i}$  might be visited at most once, as branching back into  $Q$  by taking a transition that maps to  $r$  would force  $M$  to go through all states  $q_{\text{stall}_i}$  again, which exceeds the allowed number of strokes. Note that only counter values of at most one are allowed in any run which is currently in a state in  $q_{\text{stall}_i}$  as otherwise the run will necessarily branch back into  $Q$  later on.<sup>2</sup> Especially, this is the case for  $q_{\text{stall}_0}$  which ensures that each  $n$ -turn synchronizing word has first synchronized  $Q$  into  $S$  before the first letter  $a$  is read, as otherwise  $q_{\text{stall}_0}$  is reached with a counter value greater than 1, or  $M$  has already made a turn in  $Q$  and hence cannot reach  $q_{\text{sync}}$  anymore.

In the construction above, for odd  $n$ , each run enters the synchronizing state with an empty stack (\*). For even  $n$ , each run enters the synchronizing state with a counter value of 1. The visibly condition, or more precisely very visibly condition as we are considering DVCA's, tells us that at each time while reading a synchronizing word, the stack content of every run is identical. In particular, this is the case at the point when the last state enters the synchronizing state and, hence, any  $n$ -turn synchronizing word for  $M$  is a synchronizing word in both the arbitrary and the same stack models. ◀

By observing that in the empty stack model allowing  $n$  even turns is as good as allowing  $(n - 1)$  turns, essentially (\*) from the previous proof yields the next result.

▶ **Corollary 14.** *For every fixed  $n \in \mathbb{N}$ , with  $n > 0$ , the problem  $n$ -TURN-SYNC-DVCA-EMPTY is PSPACE-hard.*

▶ **Corollary 15.** *For every fixed  $n \in \mathbb{N}$ , with  $n > 0$ , the problems  $n$ -TURN-SYNC-DVPDA and  $n$ -TURN-SYNC-DVVPDA in the empty, same, and arbitrary stack models are PSPACE-hard.*

▶ **Theorem 16.**  $0$ -TURN-SYNC-DVPDA-EMPTY  $\equiv$  DFA-SYNC.

**Proof.** The visibly condition and the fact that we can only synchronize with an empty stack mean that we cannot read any letter from  $\Sigma_{\text{call}}$ , hence we cannot use the stack at all. Delete (a) all transitions with a symbol from  $\Sigma_{\text{call}}$  and (b) all transitions with a symbol from  $\Sigma_{\text{ret}}$  and a non-empty stack. Then, assigning the elements in  $\Sigma_{\text{ret}}$  to  $\Sigma_{\text{int}}$  gives us a DFA. ◀

The next result is obtained by a reduction from DFA-SYNC-FROM-SUBSET.

▶ **Theorem 17.** *The problems  $0$ -TURN-SYNC-DVCA- $\{SAME, ARB\}$  are PSPACE-hard.*

▶ **Corollary 18.** *The problems  $0$ -TURN-SYNC-DVVPDA- $\{SAME, ARB\}$ , and  $0$ -TURN-SYNC-DVPDA- $\{SAME, ARB\}$  are PSPACE-hard.*

## 5 (Non-)Tight Upper Bounds

In this section, we will prove that at least all considered problems are decidable (in contrast to non-visibly DPDAs and DCAs, see [18]) by giving exponential time upper bounds. We will also give some tight PSPACE upper bounds for some PSPACE-hard problems discussed in the previous section, but for other previously discussed problems, a gap between PSPACE-hardness and membership in EXPTIME remains.

<sup>2</sup> In some states, such as  $q_{\text{stall}_n}$  for even  $n$ , it is simply impossible to have a higher counter value.

► **Theorem 19.** *All problems listed in Table 1 are in EXPTIME.*

**Proof.** We show the claim explicitly for SYNC-DVPDA-SAME, SYNC-DVPDA-ARB,  $n$ -TURN-SYNC-DVPDA-EMPTY,  $n$ -TURN-SYNC-DVPDA-SAME, and  $n$ -TURN-SYNC-DVPDA-ARB. The other results follow by inclusion of automata classes.

Let  $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$  be a DVPDA. We construct from  $M$  the  $|Q|$ -fold product DVPDA  $M^{|Q|}$  with state set  $Q^{|Q|}$ , consisting of  $|Q|$ -tuples of states, and alphabet  $\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}$ . Since  $M$  is a DVPDA, for every word  $w \in (\Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}})^*$ , the stack heights on runs starting in different states in  $Q$  is equal at every position in  $w$ . Hence, we can multiply the stacks to obtain the stack alphabet  $\Gamma^{|Q|}$  for  $M^{|Q|}$ . For the transition function  $\delta^{|Q|}$  (split up into  $\delta_c^{|Q|} \cup \delta_i^{|Q|} \cup \delta_r^{|Q|}$ ) of  $M^{|Q|}$ , we simulate  $\delta$  independently on every state in an  $|Q|$ -tuple, i.e., for  $(q_1, q_2, \dots, q_n) \in Q^{|Q|}$  and letters  $\sigma_c \in \Sigma_{\text{call}}, \sigma_i \in \Sigma_{\text{int}}, \sigma_r \in \Sigma_{\text{ret}}$ , we set

- $\delta_c^{|Q|}((q_1, q_2, \dots, q_n), \sigma_c) = ((q'_1, q'_2, \dots, q'_n), (\gamma_1, \gamma_2, \dots, \gamma_n))$  if  $\delta(q_j, \sigma_c) = (q'_j, \gamma_j)$  for  $j \in [n]$ ;
- $\delta_i^{|Q|}((q_1, q_2, \dots, q_n), \sigma_i) = (\delta(q_1, \sigma_i), \delta(q_2, \sigma_i), \dots, \delta(q_n, \sigma_i))$ ;
- $\delta_r^{|Q|}((q_1, q_2, \dots, q_n), \sigma_r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = (\delta(q_1, \sigma_r, \gamma_1), \delta(q_2, \sigma_r, \gamma_2), \dots, \delta(q_n, \sigma_r, \gamma_n))$ .

The bottom symbol of the stack is the  $|Q|$ -tuple  $(\perp, \perp, \dots, \perp)$ . Let  $p_1, p_2, \dots, p_n$  be an enumeration of the states in  $Q$  and set  $(p_1, p_2, \dots, p_n)$  as the start state of  $M^{|Q|}$ .

For SYNC-DVPDA-ARB, set  $\{(q, q, \dots, q) \in Q^{|Q|} \mid q \in Q\}$  as the final states for  $M^{|Q|}$ . Clearly, for SYNC-DVPDA-ARB,  $M^{|Q|}$  is a DVPDA and the words accepted by  $M^{|Q|}$  are precisely the synchronizing words for  $M$  in the arbitrary stack model. As the emptiness problem can be decided for visibly push-down automata in time polynomial in the size of the automaton [2], the claim follows observing that  $M^{|Q|}$  is exponentially larger than  $M$ .

For SYNC-DVPDA-SAME, we produce a DVPDA  $M_{\text{same}}^{|Q|}$  by enhancing the automaton  $M^{|Q|}$  with three additional states  $q_{\text{check}}, q_{\text{fin}}$ , and  $q_{\text{fail}}$  and an additional new return letter  $r$  and set  $q_{\text{fin}}$  as the single accepting state of  $M_{\text{same}}^{|Q|}$ , while the start state coincides with the one of  $M^{|Q|}$ . For states  $(q_1, q_2, \dots, q_n) \in Q^{|Q|}$ , we set  $\delta_r^{|Q|}((q_1, q_2, \dots, q_n), r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = q_{\text{check}}$  if  $q_i = q_j$  and  $\gamma_i = \gamma_j, \gamma_i \neq \perp$  for all  $i, j \in [n]$ . We set  $\delta_r^{|Q|}((q_1, q_2, \dots, q_n), r, (\perp, \perp, \dots, \perp)) = q_{\text{fin}}$  if  $q_i = q_j$  for all  $i, j \in [n]$ . For all other cases, we map with  $r$  to  $q_{\text{fail}}$ . We let the transitions for  $q_{\text{fail}}$  be defined such that  $q_{\text{fail}}$  is a non-accepting trap state for all alphabet symbols. For  $q_{\text{check}}$ , we set  $\delta_r^{|Q|}(q_{\text{check}}, r, (\gamma_1, \gamma_2, \dots, \gamma_n)) = q_{\text{check}}$  if  $\gamma_i = \gamma_j$  for  $i, j \in [n]$ . Further, we set  $\delta_r^{|Q|}(q_{\text{check}}, r, (\perp, \perp, \dots, \perp)) = q_{\text{fin}}$  and map with  $r$  to  $q_{\text{fail}}$  in all other cases. The state  $q_{\text{check}}$  also maps to  $q_{\text{fail}}$  with all input symbols other than  $r$ . We let the transitions for  $q_{\text{fin}}$  be defined such that  $q_{\text{fin}}$  is an accepting trap state for all alphabet symbols.

Clearly, for SYNC-DVPDA-SAME  $M_{\text{same}}^{|Q|}$  is a DVPDA and the words accepted by  $M_{\text{same}}^{|Q|}$  are precisely the synchronizing words for  $M$  in the same stack model, potentially prolonged by a sequence of  $r$ 's, as the single accepting state  $q_{\text{fin}}$  can only be reached from a state in  $Q^{|Q|}$  where the states are synchronized and the stack content is identical for each run (which is checked in the state  $q_{\text{check}}$ ). As the size of  $M_{\text{same}}^{|Q|}$  is exponential in the size of  $M$ , we get the claimed result as in the previous case.

For the  $n$ -TURN synchronization problems, we have to modify the previous construction by adding a stroke counter similar as in the proof of Theorem 13. ◀

► **Remark 20.** It cannot be expected to show PSPACE-membership of synchronization problems concerning DVPDAs using a  $|Q|$ -fold product DVPDA, as the resulting automata is exponentially large in the size of the DVPDA that is to be synchronized, as the emptiness problem for DVPDAs is P-complete [30]. Rather, one would need a separate membership proof. We conjecture that a PSPACE-membership proof similar to the one for the short synchronizing word problem presented in [14] can be obtained if exponential upper bounds for the length of shortest synchronizing words for DVPDAs in the respective models can be

obtained. For the empty stack model, an exponential upper bound on the length of a shortest synchronizing word should follow by applying analogous arguments as in [14, Theorem 6]. For the same and arbitrary stack model, the question is open as we cannot reduce the problem to considering pairs like in the empty stack model.

► **Theorem 21.** *The problems 0-TURN-SYNC- $\{DVPDA, DVVPDA, DVCA\}$ -SAME are in PSPACE.*

**Proof sketch.** Let  $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}} \cup \Sigma_{\text{ret}}, \Gamma, \delta, \perp)$  be a DVPDA. For the same stack model, the 0-turn condition forbids us to put in simultaneous runs different letters on the stack at any time while reading a synchronizing word, as we cannot exchange symbols on the stack with visible PDAs. Note that this is a dynamic runtime-behavior and does not imply that  $M$  is necessarily very visibly. Further, the 0-turn and visibility condition enforce that at each step the next transition does not depend on the stack content if the symbol on top of the stack is not  $\perp$ . Hence, we can construct from  $M$  a  $|Q|$ -fold DFA (with a state set exponential in the size of  $|Q|$ ) in a similar way as in the proof of Theorem 19 by neglecting the stack as nothing is ever popped from the stack. As the emptiness problem for DFAs can be solved in NLOGSPACE, the claim follows with Savitch's famous theorem stating that NPSpace = PSPACE [35].<sup>3</sup> ◀

► **Corollary 22.** *SYNC-DVPDA-SAME-NORETURN, SYNC-DVPDA-SAME are in PSPACE.*

► **Theorem 23.** *The problems 0-TURN-SYNC- $\{DVPDA, DVVPDA, DVCA\}$ -ARB, and 1-TURN-SYNC-DVCA- $\{EMPTY, SAME, ARB\}$  are in PSPACE.*

**Proof.** The claim follows from [18, Theorem 16 & 17] by inclusion of automata classes. ◀

## 6 Sequential Transducers

In [18], the concept of trace-synchronizing a sequential transducer has been introduced. We want to extend this concept to sequential transducers showing some kind of *visible behavior* regarding their output, inspired by the predetermined stack height behavior of DVPDAs. We call  $T = (Q, \Sigma, \Gamma, q_0, \delta, F)$  a *sequential transducer* (ST for short) if  $Q$  is a finite set of states,  $\Sigma$  is an input alphabet,  $\Gamma$  is an output alphabet,  $q_0$  is the start state,  $\delta: Q \times \Sigma \rightarrow Q \times \Gamma^*$  is a total transition function, and  $F$  collects the final states. We generalize  $\delta$  from input letters to words by concatenating the produced outputs.  $T$  is called a *visibly sequential transducer* (VST for short) [or *very visibly sequential transducer* (VVST for short)] if for each  $\sigma \in \Sigma$  and for all  $q_1, q_2 \in Q$  and  $\gamma_1, \gamma_2 \in \Gamma^*$ , it holds that  $\delta(q_1, \sigma) = (q'_1, \gamma_1)$  and  $\delta(q_2, \sigma) = (q'_2, \gamma_2)$  implies that  $|\gamma_1| = |\gamma_2|$  [or that  $\gamma_1 = \gamma_2$ , respectively]. A VVST  $T$  is thereby computing the same homomorphism  $h_T$ , regardless of which states are chosen as start and final states (\*). Hence, if  $A_T$  is the underlying DFA (ignoring any outputs), then  $h_T(\mathcal{L}(A_T)) \subseteq \Gamma^*$  describes the language of all possible output of  $T$ . By Nivat's theorem [29], a language family is a full trio iff it is closed under VVST and inverse homomorphisms. Our considerations also show that a language family is a full trio iff it is closed under VVST and inverse VVST mappings.

We say that a word  $w$  *trace-synchronizes* a sequential transducer  $T$  if, for all states  $p, q \in Q$ ,  $\delta(p, w) = \delta(q, w)$ , i.e., a synchronizing state is reached, producing identical output. Notice that from the viewpoint of trace-synchronization, we do not assume that a VVST has only one state.

<sup>3</sup> Here, a smaller powerset-construction would also work but, for simplicity, we stuck with the introduced  $|Q|$ -fold product construction.

► **Definition 24** (TRACE-SYNC-TRANSDUCER).

Given: Sequential transducer  $T = (Q, \Sigma, \Gamma, \delta)$ .

Question: Does there exist a word  $w \in \Sigma^*$  that trace-synchronizes  $T$ ?

We define TRACE-SYNC-VST and TRACE-SYNC-VVST by considering a VST, respectively VVST. In contrast to the undecidability of TRACE-SYNC-TRANSDUCER [18], we get the following results for trace-synchronizing VST and VVST from previous results.

► **Theorem 25.** *TRACE-SYNC-VST is PSPACE-complete.*

**Proof.** First, observe that there is a straight reduction from the problem SYNC-DVPDA-SAME-NORETURN to TRACE-SYNC-VST as the input DVPDAs to the problem SYNC-DVPDA-SAME-NORETURN have no return letters and, hence, the stack is basically a write only tape. Further, as the remaining alphabet is partitioned into letters in  $\Sigma_{\text{call}}$ , which write precisely one symbol on the stack, and into letters in  $\Sigma_{\text{int}}$ , writing nothing on the stack, the visibly condition is satisfied when interpreting the DVPDA with  $\Sigma_{\text{ret}} = \emptyset$  as a VST.

There is also a straight reduction from TRACE-SYNC-VST to SYNC-DVPDA-SAME-NORETURN as follows. For a VST  $T = (Q, \Sigma, \Gamma, \delta)$ , we construct a DVPDA  $M = (Q, \Sigma_{\text{call}} \cup \Sigma_{\text{int}}, \Gamma', \delta)$  with  $\Sigma_{\text{ret}} = \emptyset$  by introducing for each  $\sigma \in \Sigma$  a new alphabet  $\Sigma_{\sigma} = \{w \in \Gamma^* \mid \exists q, q' \in Q: \delta(q, \sigma) = (q', w)\}$ . Observe that  $\Sigma_{\sigma}$  is either  $\{\epsilon\}$  or contains only words of the same length. By setting  $\Sigma_{\text{int}} = \{\sigma \in \Sigma \mid \Sigma_{\sigma} = \{\epsilon\}\}$ ,  $\Sigma_{\text{call}} = \{\sigma \in \Sigma \mid \Sigma_{\sigma} \neq \{\epsilon\}\}$ ,  $\Gamma' = \bigcup_{\sigma \in \Sigma} (\Sigma_{\sigma} \setminus \{\epsilon\})$ , and interpreting the output sequence  $w \in \Gamma^*$  produced by  $\delta$  as the single stack symbol in  $\Gamma'$ . ◀

Yet, by Observation (\*), we inherit from SYNC-DFA the following algorithmic result.

► **Theorem 26.** *TRACE-SYNC-VVST is in P.*

## 7 Discussion

Our results concerning DVPDAs and sub-classes thereof are summarized in Table 1. While all problems listed in the table are contained in EXPTIME, the table lists several problems for which their known complexity status still contains a gap between PSPACE-hardness lower bounds and EXPTIME upper bounds. Presumably, their precise complexity status is closely related to upper bounds on the length of synchronizing words which we want to consider in the near future. One of the questions which could be solved in this work is if there is a difference between the complexity of synchronization in the same stack model and synchronization in the arbitrary stack model. While for general DPDA, DCA, and sub-classes thereof, see [18], these two models admitted synchronization problems with the same complexity, here we observed that these models can differ significantly. While the focus of this work is on determining the complexity status of synchronizability for different models of automata, an obvious question for future research is the complexity status of closely related, and well understood questions in the realm of DFAs, such as the problem of shortest synchronizing word, subset synchronization, synchronization into a subset, and careful synchronization.

Here is one subtlety that comes with shortest synchronizing words: While for finding synchronizing words of length at most  $k$  for DFAs, it does not matter if the number  $k$  is given in unary or in binary due to the known cubic upper bounds on the lengths of shortest synchronizing words, this will make a difference in other models where such polynomial length bounds are unknown. More precisely, for instance with DVPDAs, it is rather obvious that with a unary length bound  $k$ , the problem becomes NP-complete, while the status is unclear for binary length bounds. As there is no general polynomial upper bound on the length of

shortest synchronizing words for VPDAs, they might be of exponential length. Hence, we do not get membership in PSPACE easily, not even for synchronization models concerning DVPDA for which general synchronizability is solvable in P, as it might be necessary to store the whole word on the stack in order to test its synchronization effects.

---

## References

- 1 Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. [https://www.jalc.de/issues/2019/issue\\_24\\_2-4/content.html](https://www.jalc.de/issues/2019/issue_24_2-4/content.html). Accessed: 10/1/2020.
- 2 Rajeev Alur and P. Madhusudan. Visibly Pushdown Languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.
- 3 Rajeev Alur and P. Madhusudan. Adding Nesting Structure to Words. *J. ACM*, 56(3):16:1–16:43, 2009.
- 4 Marcelo Arenas, Pablo Barceló, and Leonid Libkin. Regular Languages of Nested Words: Fixed Points, Automata, and Synchronization. *Theory of Computing Systems*, 49(3):639–670, 2011.
- 5 Parvaneh Babari, Karin Quaas, and Mahsa Shirmohammadi. Synchronizing Data Words for Register Automata. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 6 Vince Bárány, Christof Löding, and Olivier Serre. Regularity Problems for Visibly Pushdown Languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.
- 7 Marie-Pierre Béal and Dominique Perrin. *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2016.
- 8 Jean Berstel. *Transductions and Context-Free Languages*, volume 38 of *Teubner Studienbücher: Informatik*. Teubner, 1979.
- 9 Benedikt Bollig. One-Counter Automata with Counter Observability. In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, Proceedings*, volume 65 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 10 Olivier Carton. The Growth Ratio of Synchronous Rational Relations is Unique. *Theoretical Computer Science*, 376(1-2):52–59, 2007.
- 11 Didier Caucal. Synchronization of Pushdown Automata. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2006.
- 12 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovensk*, 14(3):208–215, 1964.
- 13 Ján Černý. A Note on Homogeneous Experiments with Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132, 2019.
- 14 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing Automata over Nested Words. *Journal of Automata, Languages and Combinatorics*, 24(2-4):219–251, 2019.
- 15 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing Words for Weighted and Timed Automata. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 121–132, 2014.
- 16 David Eppstein. Reset Sequences for Monotonic Automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.

- 17 Henning Fernau and Petra Wolf. Synchronization of Deterministic Visibly Push-Down Automata. *CoRR*, abs/2005.01374, 2020. [arXiv:2005.01374](https://arxiv.org/abs/2005.01374).
- 18 Henning Fernau, Petra Wolf, and Tomoyuki Yamakami. Synchronizing Deterministic Push-Down Automata Can Be Really Hard. *CoRR*, abs/2005.01381, 2020. An extended abstract is accepted at MFCS 2020. [arXiv:2005.01381](https://arxiv.org/abs/2005.01381).
- 19 Seymour Ginsburg. *The mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.
- 20 Seymour Ginsburg and Edwin H Spanier. Finite-Turn Pushdown Automata. *SIAM Journal on Control*, 4(3):429–453, 1966.
- 21 Michael Hahn, Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. Visibly Counter Languages and the Structure of  $NC^1$ . In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 384–394. Springer, 2015.
- 22 Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. On Distinguishing  $NC^1$  and NL. In Igor Potapov, editor, *Developments in Language Theory - 19th International Conference, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2015.
- 23 Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. Visibly Counter Languages and Constant Depth Circuits. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *LIPICs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
- 24 Michael Ludwig. *Tree-Structured Problems and Parallel Computation*. PhD thesis, University of Tübingen, Germany, 2019. URL: <https://publikationen.uni-tuebingen.de/xmlui/handle/10900/85960/>.
- 25 Pavel Martyugin. Computational Complexity of Certain Problems Related to Carefully Synchronizing Words for Partial Automata and Directing Words for Nondeterministic Automata. *Theory of Computing Systems*, 54(2):293–304, 2014.
- 26 Pavel V. Martyugin. Synchronization of Automata with One Undefined or Ambiguous Transition. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2012.
- 27 Kurt Mehlhorn. Pebbling Mountain Ranges and its Application of DCFL-Recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordwijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.
- 28 Eitatsu Mikami and Tomoyuki Yamakami. Synchronizing Pushdown Automata and Reset Words, 2020. An article appeared in Japanese as Technical Report of The Institute of Electronics, Information and Communication Engineers, COMP2019-54(2020-03), pp. 57–63.
- 29 Maurice Nivat. Transductions des langages de Chomsky. *Ann. Inst. Fourier, Grenoble*, 18:339–456, 1968.
- 30 Alexander Okhotin and Kai Salomaa. Complexity of Input-Driven Pushdown Automata. *SIGACT News*, 45(2):47–67, 2014.
- 31 I. K. Rystsov. On Minimizing the Length of Synchronizing Words for Finite Automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci., 1980. (in Russian).
- 32 I. K. Rystsov. Polynomial Complete Problems in Automata Theory. *Information Processing Letters*, 16(3):147–151, 1983.
- 33 Jacques Sakarovitch. *Éléments de Théorie des Automates*. Vuibert informatique, 2003.
- 34 Sven Sandberg. Homing and Synchronizing Sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.

- 35 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- 36 Mahsa Shirmohammadi. *Qualitative Analysis of Synchronizing Probabilistic Systems. (Analyse qualitative des systèmes probabilistes synchronisants)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: <https://tel.archives-ouvertes.fr/tel-01153942>.
- 37 Yaroslav Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):367–373, 2019.
- 38 Jirí Srba. Beyond Language Equivalence on Visibly Pushdown Automata. *Logical Methods in Computer Science*, 5(1), 2009.
- 39 Peter H. Starke. Eine Bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik (J. Inf. Process. Cybern.)*, 2(4):257–259, 1966.
- 40 Peter H. Starke. A Remark About Homogeneous Experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137, 2019.
- 41 Marek Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPICs*, pages 56:1–56:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 42 Leslie G. Valiant. *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, Coventry, UK, 1973. URL: <http://wrap.warwick.ac.uk/34701/>.
- 43 Mikhail V. Volkov. Synchronizing Automata and the Černý Conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.



# Synthesis from Weighted Specifications with Partial Domains over Finite Words

**Emmanuel Filiot**

Université libre de Bruxelles, Belgium  
efiliot@ulb.ac.be

**Christof Löding**

RWTH Aachen University, Germany  
loeding@cs.rwth-aachen.de

**Sarah Winter**

Université libre de Bruxelles, Belgium  
swinter@ulb.ac.be

---

## Abstract

In this paper, we investigate the synthesis problem of terminating reactive systems from quantitative specifications. Such systems are modeled as finite transducers whose executions are represented as finite words in  $(\Sigma_{\text{i}} \times \Sigma_{\text{o}})^*$ , where  $\Sigma_{\text{i}}$ ,  $\Sigma_{\text{o}}$  are finite sets of input and output symbols, respectively. A weighted specification  $S$  assigns a rational value (or  $-\infty$ ) to words in  $(\Sigma_{\text{i}} \times \Sigma_{\text{o}})^*$ , and we consider three kinds of objectives for synthesis, namely threshold objectives where the system's executions are required to be above some given threshold, best-value and approximate objectives where the system is required to perform as best as it can by providing output symbols that yield the best value and  $\varepsilon$ -best value respectively w.r.t.  $S$ . We establish a landscape of decidability results for these three objectives and weighted specifications with partial domain over finite words given by deterministic weighted automata equipped with sum, discounted-sum and average measures. The resulting objectives are not regular in general and we develop an infinite game framework to solve the corresponding synthesis problems, namely the class of (weighted) critical prefix games.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic and verification; Theory of computation  $\rightarrow$  Transducers; Theory of computation  $\rightarrow$  Quantitative automata

**Keywords and phrases** synthesis, weighted games, weighted automata on finite words

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.46

**Funding** *Emmanuel Filiot*: This work is partially supported by the MIS project F451019F (F.R.S.-FNRS). Emmanuel Filiot is a research associate at F.R.S.-FNRS.

## 1 Introduction

**Reactive synthesis.** The goal of automatic synthesis is to automatically construct programs from specifications of correct pairs of input and output. The goal is to liberate the developer from low-level implementation details, and to automatically generate programs which are correct by construction. In the automata-based approach to synthesis [14, 20], the programs to be synthesized are finite-state reactive programs, which react continuously to stimuli received from an environment. Such systems are not assumed to terminate and their executions are modeled as  $\omega$ -words in  $(\Sigma_{\text{i}}\Sigma_{\text{o}})^\omega$ , alternating between input symbols in  $\Sigma_{\text{i}}$  and output symbols in  $\Sigma_{\text{o}}$ . Specifications of such systems are then languages  $S \subseteq (\Sigma_{\text{i}}\Sigma_{\text{o}})^\omega$  representing the set of acceptable executions. The synthesis problem asks to check whether there exists a total synchronous<sup>1</sup> function  $f: \Sigma_{\text{i}}^\omega \rightarrow \Sigma_{\text{o}}^\omega$  such that for all input sequences  $u = i_0i_1\dots$ , there exists

---

<sup>1</sup>  $f: \Sigma_{\text{i}}^\omega \rightarrow \Sigma_{\text{o}}^\omega$  is synchronous if it is induced by a strategy  $s: \Sigma_{\text{i}}^+ \rightarrow \Sigma_{\text{o}}$  in the sense that  $f(i_0i_1\dots) = s(i_0)s(i_0i_1)s(i_0i_1i_2)\dots$  for all  $i_0i_1\dots \in \Sigma_{\text{i}}^\omega$



an output sequence  $v = o_0o_1 \dots$  such that  $f(u) = v$  and the convolution  $u \otimes v = i_0o_0i_1o_1 \dots$  belongs to  $S$ . The function  $f$  is called a realizer of  $S$ . Automatic synthesis of non-terminating reactive systems has first been introduced by Church [19], and a first solution has been given by Büchi and Landweber [14] when the specification  $S$  is  $\omega$ -regular. In this setting, when a realizer exists, there is always one which can be computed by a finite-state sequential transducer, a finite-state automaton which alternates between reading one input symbol and producing one output symbol. This result has sparked much further work to make synthesis feasible in practice, see e.g., [31, 27, 7]. The synthesis problem is classically modeled as an infinite-duration game on a graph, played by two players, alternatively picking input and output symbols. One player, representing the system, must enforce an objective that corresponds to the specification. Finite-memory winning strategies are in turn systems that realize the specification. This game metaphor has triggered a lot of research on graph games [20, Chapter 27]. There has also been a recent effort to increase the quality of the automatically generated systems by enhancing Boolean specifications with quantitative constraints, e.g., [5, 16, 12, 2]. This has also triggered a lot of research on quantitative extensions of infinite-duration games, for example mean-payoff, energy, and discounted-sum games, see, e.g., [24, 34, 22, 10, 11, 4, 29].

**Partial-domain specifications.** In the classical formulation of the synthesis problem, it is required that a realizer  $f$  meets the specification *for all* possible input sequences. In particular, if there is a single input sequence  $u$  such that  $u \otimes v \notin S$  for all output sequences  $v$ , then  $S$  admits no realizer. In other words, when the *domain* of  $S$  is partial, then  $S$  is unrealizable. Formally, the domain of  $S$  is  $\text{dom}(S) = \{u \in \Sigma_{\ddagger}^{\omega} \mid \exists v: u \otimes v \in S\}$ . As noticed recently and independently in [1], asking that the realizer meets the specification for all input sequences is often too strong and a more realistic setting is to make some assumptions on the environment's behaviour, namely, that the environment plays an input sequence in the domain of the specification. This problem is called *good-enough synthesis* in [1] and can be formulated as follows: given a specification  $S$ , check whether there exists a *partial* synchronous function  $f: \Sigma_{\ddagger}^{\omega} \rightarrow \Sigma_{\circ}^{\omega}$  whose domain is  $\text{dom}(S)$ , and such that for all input sequence  $u \in \text{dom}(S) = \text{dom}(f)$ ,  $u \otimes f(u) \in S$ . Decidability of the latter problem is entailed by decidability of the classical synthesis problem when the specification formalism used to describe  $S$  is closed under expressing the assumption that the environment provides inputs in  $\text{dom}(S)$ . It is the case for instance when  $S$  is  $\omega$ -regular, because the specification  $S \cup \overline{\text{dom}(S)} \otimes \Sigma_{\circ}^{\omega}$  has total domain and is effectively  $\omega$ -regular. [1] investigates the more challenging setting of  $S$  being expressed by a multi-valued (in contrast to Boolean) LTL logic. More generally, there is a series of works on solving games under assumptions on the behaviour of the environment [18, 6, 30, 21, 13, 2].

**Our setting: Partial-domain weighted specifications.** In this paper, motivated by the line of work on quantitative extensions of synthesis and the latter more realistic setting of partial-domain specifications, we investigate synthesis problems from partial-domain weighted specifications (hereafter just called weighted specifications). We conduct this investigation in the setting of *terminating reactive systems*, and accordingly our specifications are over finite words. Formally, a specification is a mapping  $S: (\Sigma_{\ddagger}.\Sigma_{\circ})^* \rightarrow \mathbb{Q} \cup \{-\infty\}$ . The domain  $\text{dom}(S)$  of  $S$  is defined as all the input sequences  $u \in \Sigma_{\ddagger}^*$  such that  $S(u \otimes v) \in \mathbb{Q}$  for some  $v \in \Sigma_{\circ}^*$ . We consider three quantitative synthesis problems, which all consists in checking whether there exists a function  $f$  computable by a finite transducer such that  $\text{dom}(f) = \text{dom}(S)$  and which satisfies respectively the following conditions:

■ **Table 1** Complexity results for weighted specifications. Here, D stands for decidable, the suffix -c for complete,  $\lambda$  for discount factor, and  $n$  for a natural number.

Problem \ Spec	Sum-automata	Avg-automata	Dsum-automata
strict threshold	$\text{NP} \cap \text{coNP}$	$\text{NP} \cap \text{coNP}$	NP
non-strict threshold	$\text{NP} \cap \text{coNP}$	$\text{NP} \cap \text{coNP}$	$\text{NP} \cap \text{coNP}$
best-value	P <sub>TIME</sub> [3]	P <sub>TIME</sub> [3]	$\text{NP} \cap \text{coNP}$
strict approximate	EXP <sub>TIME-c</sub> [26]	D	NEXP <sub>TIME</sub> for $\lambda = 1/n$
non-strict approx.	EXP <sub>TIME-c</sub> [26]	D	EXP <sub>TIME</sub> for $\lambda = 1/n$

- for all  $u \in \text{dom}(S)$  it holds that  $S(u \otimes f(u)) \triangleright t$  for a given threshold  $t \in \mathbb{Q}$  and  $\triangleright \in \{>, \geq\}$ , called *threshold synthesis*, or
- $S(u \otimes f(u)) = \text{bestVal}_S(u)$ , that is, the maximal value that can be achieved for the input  $u$ , i.e.,  $\text{bestVal}_S(u) = \sup\{S(u \otimes v) \mid v \in \Sigma_o^*\}$ , called *best-value synthesis*, or
- $\text{bestVal}_S(u) - S(u \otimes f(u)) \triangleleft r$  for a given threshold  $r \in \mathbb{Q}$  and  $\triangleleft \in \{<, \leq\}$ , called *approximate synthesis*.

Following the game metaphor explained before, those quantitative synthesis problems can be formulated as two-player games in which Adam (environment) and Eve (system) alternatively pick symbols in  $\Sigma_{\mathfrak{a}}$  and  $\Sigma_{\mathfrak{o}}$  respectively. Additionally, Adam has the power to stop the game. If it does not, then Eve wins the game. Otherwise, a finite play spells a word  $u \otimes v$ . For the Boolean synthesis problem, Eve has won if either  $u \notin \text{dom}(S)$  where  $S$  is the specification, or  $u \otimes v \in S$ . Additionally, for the threshold synthesis problem, the value  $S(u \otimes v)$  must be greater than the given threshold; for the best-value synthesis problem, it must be equal to  $\text{bestVal}_S(u)$  and for approximate synthesis it must be  $r$ -close to  $\text{bestVal}_S(u)$ .

**Contributions.** Our main contribution is a clear picture about decidability of threshold synthesis, best-value synthesis and approximate synthesis for weighted specifications over finite words defined by deterministic weighted finite automata [23], equipped with either sum, average or discounted-sum measure. Such automata extend finite automata with integer weights on their transitions, computing a value through a payoff function that combines those integers, with sum, average, or discounted-sum. The results (presented in Section 4) are summarized in Table 1. We also give an application of our results to the decidability of quantitative extensions of the Church synthesis problem over infinite words, for some classes of weighted safety specifications, which intuitively require that all prefixes satisfy a quantitative requirement (being above a threshold, equal to the best-value, or close to it).

As we explain in the related works section, some of our results are obtained via reduction to solving known quantitative games or to the notions of  $r$ -regret determinization for weighted automata. We develop new techniques to solve the strict threshold synthesis problem for discounted-sum specifications in NP (Theorem 9), the best-value synthesis problem for discounted-sum specifications in  $\text{NP} \cap \text{coNP}$  (Theorem 12) and approximate synthesis for average specifications (Theorem 13), which are to the best of our knowledge new results.

Moreover, as our main tool to obtain our synthesis results, we introduce in Section 3 a new kind of (weighted) games called *critical prefix games* tailored to handle weighted specifications with *partial* domain of *finite* words. We believe these kind of games are interesting on their own and are described below in more detail.

**Critical prefix games.** Following the classical game metaphor of synthesis, we design weighted games into which some of our synthesis problems can be directly encoded. Those games still have infinite-duration, but account for the fact that specifications are on finite words and have partial domains. In particular, the quantitative constraints must be checked only for play prefixes that correspond to input words of the environment which are in the domain of the specification. So, a critical prefix game is defined as a two-player turn-based weighted game with some of the vertices being declared as critical. When the play enters a critical vertex, a quantitative requirement must be fulfilled, otherwise Eve loses. For instance, critical prefix *threshold* games require that the payoff value when entering a critical vertex is at least or above a certain threshold. We show that these threshold games are all decidable for sum, average, and discounted-sum payoffs, see Theorems 3 and 4. For solving approximate average synthesis, we use a reduction to critical prefix energy games of imperfect information starting with fixed initial credit (the energy level must be at least zero whenever the play is in a critical vertex). Without critical vertices (where the energy level must be at least zero all the time) these games are known to be decidable [22]. We show that adding critical vertices makes these games undecidable, in general, see Theorem 7. However, a large subclass of imperfect information critical prefix energy games, sufficient for our synthesis problems, is shown to be decidable, see Theorem 8.

**Domain-safe weighted specifications.** Most of our quantitative synthesis problems reduce to two-player games. While we need games of different natures, they all model the fact that Eve constructs a run of the (deterministic) automaton, given the input symbols provided by Adam so far. By choosing outputs, Eve must make sure that this run is accepting whenever the input word played by Adam so far is in the domain of  $S$ . Otherwise Adam can stop and Eve loses. While this condition can be encoded in the game by enriching the vertices with subsets of states (in which Eve could have been by choosing alternative output symbols), this would result in an exponential blow-up of the game. We instead show that the weighted automaton can be preprocessed in polynomial-time into a so called domain-safe automaton, in which there is no need to monitor the input domain when playing, see Theorem 2.

**Related works.** Boolean synthesis problems for finite words have been considered in [33, 32] where the specification is given as an LTL formula over finite traces. In the quantitative setting, it has also been considered in [25] for weighted specifications given by deterministic weighted automata. In these works however, it is the role of Eve to eventually stop the game. While this makes sense for reachability objectives and planning problems, this setting does not accurately model a synthesis scenario where the system has no control over the provided input sequence. Our setting is different and needs new technical developments.

Threshold problems in quantitative infinite-duration two-player games with discounted- and mean-payoff measures are known to be solvable in  $\text{NP} \cap \text{coNP}$  [4, 34]. Our threshold synthesis problems all directly reduce to critical prefix threshold games with corresponding payoff functions. The latter games, for sum and average, are shown to reduce to mean-payoff games, so our  $\text{NP} \cap \text{coNP}$  upper-bound follows from [34]. For critical prefix discounted-sum games with a non-strict threshold, we show a polynomial time reduction to infinite-duration discounted sum games and hence our result follows from [4]. Such a reduction fails for a strict threshold and we develop new techniques to solve critical prefix discounted-sum games with strict threshold, by first showing that memoryless strategies suffice for Eve to win, and then by showing how to check in PTIME whether a memoryless strategy is winning for Eve. The latter result actually shows how to test in PTIME whether there exists,

in a weighted graph, a path from a source to a target vertex of discounted-sum greater or equal to some given threshold. This result entails that the non-emptiness problem for non-deterministic discounted-sum max-automata<sup>2</sup> is solvable in PTIME (Theorem 6). To the best of our knowledge, up to now this problem is only known to be in PSPACE for the subcase of functional discounted-sum automata [25, 9].

As we show, the best-value synthesis problems correspond to zero-regret determinization problems for non-deterministic weighted automata, i.e., deciding whether there is a non-determinism resolving strategy for Eve that guarantees the same value as the maximal value of an accepting run in the non-deterministic weighted automaton. Such a problem is in PTIME for sum-automata [3] and the average case easily reduces to the sum-case. For discounted-sum, zero-regret determinization is known to be decidable in NP for dsum-automata over infinite words [29]. We improve this bound to  $\text{NP} \cap \text{CONP}$  for finite words.

Finally, approximate synthesis corresponds to a problem known as  $r$ -regret determinization of non-deterministic weighted automata. For sum-automata, it is known to be EXPTIME-complete [26]. For average-automata, there is no immediate reduction to the sum case, because the sum value computed by an  $r$ -regret determinizer can be arbitrarily faraway from the best sum, while its averaged value remains close to the best average. Instead, we show a reduction to the new class of partial observation critical prefix energy games. For dsum-automata over infinite words, total domain and integral discount factor,  $r$ -regret determinization is known to be decidable [29]. Our setting does not directly reduce to this setting, but we use similar ideas.

## 2 Preliminaries

**Languages and relations.** Let  $\mathbb{N}$  be the set of non-negative integers. Let  $\Sigma$  be a finite alphabet. We denote by  $\Sigma^*$ , respectively  $\Sigma^\omega$ , the set of finite, respectively infinite, words over  $\Sigma$ , and  $\Sigma^+$  the set of non-empty finite words over  $\Sigma$ . The empty word is denoted by  $\varepsilon$ . A *language* over  $\Sigma$  is a set of words over  $\Sigma$ . A (binary) *relation*  $R$  is a subset of  $\Sigma_{\mathfrak{i}}^* \times \Sigma_{\mathfrak{o}}^*$ , i.e., a set of pairs of words. Its domain is the set  $\text{dom}(R) = \{u \mid \exists v: (u, v) \in R\}$ . Given a pair of words, we refer to the first (resp. second) component as input (resp. output) component, the alphabets  $\Sigma_{\mathfrak{i}}$  and  $\Sigma_{\mathfrak{o}}$  are referred to as input resp. output alphabet. We let  $\Sigma_{\mathfrak{i}\mathfrak{o}} = \Sigma_{\mathfrak{i}} \cup \Sigma_{\mathfrak{o}}$ .

**Automata.** A *nondeterministic finite state automaton (NFA)* is a tuple  $\mathcal{A} = (Q, q_i, \Sigma, \Delta, F)$ , where  $Q$  is a finite state set,  $q_i \in Q$  is the initial state,  $\Sigma$  is a finite alphabet,  $\Delta \subseteq Q \times \Sigma \times Q$  is a transition relation, and  $F \subseteq Q$  is a set of final states. A *run* of the automaton on a word  $w = a_1 \dots a_n$  is a sequence  $\rho = \tau_1 \dots \tau_n$  of transitions such that there exist  $q_0, \dots, q_n \in Q$  such that  $\tau_j = (q_{j-1}, a_j, q_j)$  for all  $j$ . A run on  $\varepsilon$  is a single state. A run is *accepting* if it begins in the initial state and ends in a final state. The *language recognized* by the automaton is defined as  $L(\mathcal{A}) = \{w \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}$ . The automaton is *deterministic (a DFA)* if  $\Delta$  is given as a partial function  $\delta: Q \times \Sigma \rightarrow Q$ .

**Transducers.** A *transducer* is a tuple  $\mathcal{T} = (Q, q_i, \Sigma_{\mathfrak{i}}, \Sigma_{\mathfrak{o}}, \delta, F)$ , where  $Q$  is a finite state set,  $q_i \in Q$  is the initial state,  $\Sigma_{\mathfrak{i}}$  and  $\Sigma_{\mathfrak{o}}$  are finite alphabets,  $\delta: (Q \times \Sigma_{\mathfrak{i}}) \rightarrow (\Sigma_{\mathfrak{o}} \times Q)$  is a transition function, and  $F \subseteq Q$  is a set of final states. A transition is also denoted as a tuple for convenience. A *run* is either a non-empty sequence of transitions  $\rho = (q_0, u_1, v_1, q_1)(q_1, u_2, v_2, q_2) \dots (q_{n-1}, u_n, v_n, q_n)$  or a single state. The *input (resp. output)* of

<sup>2</sup> i.e., checking whether there exists a word with value greater or equal to some threshold, where the value is defined by taking the max over all accepting runs.

$\rho$  is  $u = u_1 \dots u_n$  (resp.  $v = v_1 \dots v_n$ ) if  $\rho \in \Delta^+$ , both are  $\varepsilon$  if  $\rho \in Q$ . We denote by  $p \xrightarrow{u|v} q$  that there exists a run from  $p$  to  $q$  with input  $u$  and output  $v$ . A run is *accepting* if it starts in the initial and ends in a final state. The *partial function recognized* by the transducer is  $f_{\mathcal{T}}: \Sigma_{\mathfrak{I}}^* \rightarrow \Sigma_{\mathfrak{O}}^*$  defined as  $f_{\mathcal{T}}(u) = v$  if there is an accepting run of the form  $p \xrightarrow{u|v} q$ .

**Weighted automata.** Let  $n > 0$ . Given a finite sequence  $\phi = j_1 \dots j_n$  of integers, and a discount factor  $\lambda \in \mathbb{Q}$  such that  $0 < \lambda < 1$ , we define the following functions:  $\text{Sum}(\phi) = \sum_{i=1}^n j_i$ ,  $\text{Avg}(\phi) = \frac{\text{Sum}(\phi)}{n}$ ,  $\text{Dsum}(\phi) = \sum_{i=1}^n \lambda^i j_i$  if  $\phi$  is non-empty and  $\text{Sum}(\phi) = \text{Avg}(\phi) = \text{Dsum}(\phi) = 0$  otherwise. Let  $V \in \{\text{Sum}, \text{Avg}, \text{Dsum}\}$ . A *weighted V-automaton (WFA)* is a tuple  $\mathcal{A} = (Q, \Sigma, q_i, \Delta, F, \gamma)$ , where  $(Q, \Sigma, q_i, \Delta, F)$  is a classical *deterministic* finite state automaton, and  $\gamma: \delta \rightarrow \mathbb{Z}$  is a *weight function*. Its recognized language, etc., is defined as for classical finite state automata. The value  $V(\rho)$  of a run  $\rho = \tau_1 \dots \tau_n$  is defined as  $V(\gamma(\tau_1) \dots \gamma(\tau_n))$  if  $\rho$  is accepting and  $-\infty$  otherwise. The value  $\mathcal{A}(w)$  of a word  $w$  is given by the total function, called the function *recognized* by  $\mathcal{A}$ ,  $\mathcal{A}: \Sigma^* \rightarrow \mathbb{Q} \cup \{-\infty\}$  defined as  $w \mapsto V(\rho)$ , where  $\rho$  is the run of  $\mathcal{A}$  on  $w$ , that is, the value of a word is the value of its accepting run, or  $-\infty$  if there exists none.

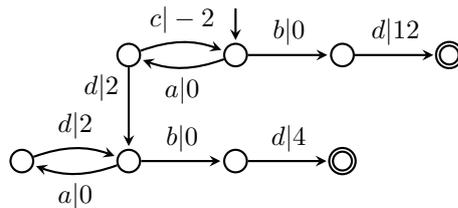
**Weighted specifications.** A *weighted specification* is a total function  $S: (\Sigma_{\mathfrak{I}}\Sigma_{\mathfrak{O}})^* \rightarrow \mathbb{Q} \cup \{-\infty\}$  recognized by a WFA  $\mathcal{A}$ . Note that by our definition,  $\mathcal{A}$  is deterministic by default. Given  $u = u_1 \dots u_n \in \Sigma_{\mathfrak{I}}^*$  and  $v = v_1 \dots v_n \in \Sigma_{\mathfrak{O}}^*$ ,  $u \otimes v$  denotes its *convolution*  $u_1 v_1 \dots u_n v_n \in (\Sigma_{\mathfrak{I}}\Sigma_{\mathfrak{O}})^*$ . We usually write  $S(u \otimes v)$  instead of  $S(u_1 v_1 \dots u_n v_n)$ . The relation (or Boolean specification) of  $S$ , denoted by  $R(S)$ , is given by the set of pairs that are mapped to a rational number, i.e.,  $R(S) = \{(u, v) \mid S(u \otimes v) > -\infty\}$ . We usually write  $u \otimes v \in S$  instead of  $(u, v) \in R(S)$ . The domain of  $S$ , denoted by  $\text{dom}(S)$ , is defined as  $\{u \in \Sigma_{\mathfrak{I}}^* \mid \exists v \in \Sigma_{\mathfrak{O}}^*: u \otimes v \in S\}$ . If a weighted specification is given by some  $V$ -automaton, we refer to it as  $V$ -specification.

**Quantitative synthesis problems.** The (Boolean) *synthesis problem* asks, given a weighted specification  $S$ , whether there exists a partial function  $f: \Sigma_{\mathfrak{I}}^* \rightarrow \Sigma_{\mathfrak{O}}^*$  defined by a transducer with  $\text{dom}(f) = \text{dom}(S)$  such that  $u \otimes f(u) \in S$  for all  $u \in \text{dom}(f)$ .

We define three quantitative synthesis problems that pose additional conditions, we only state the additions. The *threshold synthesis problem* additionally asks, given a threshold  $\nu \in \mathbb{Q}$ , and  $\triangleright \in \{>, \geq\}$ , that  $S(u \otimes f(u)) \triangleright \nu$  for all  $u \in \text{dom}(f)$ . The *best-value synthesis problem* additionally asks that  $S(u \otimes f(u)) = \text{bestVal}_S(u)$ , where  $\text{bestVal}_S(u) = \sup\{S(u \otimes v) \mid u \otimes v \in S\}$  for all  $u \in \text{dom}(f)$ . The *approximate synthesis problem* additionally asks, given a threshold  $\nu \in \mathbb{Q}$ , and  $\triangleleft \in \{<, \leq\}$ , that  $\text{bestVal}_S(u) - S(u \otimes f(u)) \triangleleft \nu$  for all  $u \in \text{dom}(f)$ .

In these settings, if such a function  $f$  exists, it is called  $S$ -*realization*, a transducer that defines  $f$  is called  $S$ -*realizer*, and is said to *implement an S-realization*. A transducer whose implemented function  $f$  only satisfies the Boolean condition is called *Boolean S-realizer*.

► **Example 1.** Let  $\Sigma_{\mathfrak{I}} = \{a, b\}$  and  $\Sigma_{\mathfrak{O}} = \{c, d\}$ , and consider the weighted specification  $S$  defined by the following automaton  $\mathcal{A}$ .



Clearly,  $S$  has a Boolean realizer (infinitely many, in fact). First, we view  $\mathcal{A}$  as a Sum-automaton. There exists a realizer that ensures a value of at least 6, for example, the transducer that always outputs  $d$ . There exists no best-value realizer. To see this, we look at the maximal values. We have  $\text{bestVal}(b) = 12$ ,  $\text{bestVal}(ab) = 10$ , and  $\text{bestVal}(a^i b) = 2i + 4$  for  $i > 1$ . The maximal value for  $ab$  is achieved with  $cd$  and the maximal value for  $aaab$  with  $dddd$ . So, the first output symbol depends on the length of the input word, which is unknown to a transducer when producing the first output symbol. However, there exists an approximate realizer for the non-strict threshold 4: the transducer that outputs  $c$  solely for the first  $a$ . The difference to the maximal value is 0 for the inputs  $b$  and  $ab$ , and 4 for all other inputs. Secondly, we view  $\mathcal{A}$  as an Avg-automaton. With the same argumentation as for Sum, it is easy to see that there exists no best-value realizer, there exists an approximate realizer for the non-strict threshold  $\frac{2}{3}$ : the transducer that outputs  $c$  solely for the first  $a$ . The difference to the maximal value is 0 for the inputs  $b$  and  $ab$ , and  $\frac{2}{i+1}$  for inputs of the form  $a^i b$  for  $i > 1$ . Note that the difference decreases with the input length unlike for Sum.

**Boolean synthesis and domain-safe automata.** The quantitative synthesis problems that we have defined, ask for Boolean realizers that additionally satisfy a quantitative condition. We start by showing that a weighted specification  $\mathcal{A}$  can be preprocessed in polynomial time such that dealing with the Boolean part becomes very simple. Basically, we remove all parts of  $\mathcal{A}$  that cannot be used by a Boolean realizer. We call the result of this preprocessing a *domain-safe weighted specification*, to be defined formally below. In Section 4 we use domain-safe specifications.

Denote by  $\text{dom}(\mathcal{A}) \subseteq \Sigma_{\mathbb{I}}^*$  the domain of the weighted specification defined by  $\mathcal{A}$ . We can easily obtain an NFA (with  $\varepsilon$ -transitions) for  $\text{dom}(\mathcal{A})$  by removing the weights and turning all transitions that are labelled by an output letter into an  $\varepsilon$ -transition. We call the resulting NFA the domain automaton of  $\mathcal{A}$ , and denote it by  $\mathcal{A}_{\text{dom}}$ . For a state  $q$  of  $\mathcal{A}$ , we denote by  $L(\mathcal{A}_{\text{dom}}, q)$  the language of  $\mathcal{A}_{\text{dom}}$  accepted by runs starting in  $q$ . An output transition  $(q, a, q')$  of  $\mathcal{A}$  is called *domain-safe* if  $L(\mathcal{A}_{\text{dom}}, q) = L(\mathcal{A}_{\text{dom}}, q')$ , i.e., it does not restrict the language of input words that can be accepted by  $\mathcal{A}_{\text{dom}}$ . Otherwise, such a transition is called *domain-unsafe*. We call a weighted specification  $\mathcal{A}$  *domain-safe* if it is trim, i.e., all states are accessible and co-accessible, and all its output transitions are domain-safe.

A transducer that produces an input/output pair whose run in  $\mathcal{A}$  uses a domain-unsafe transition of  $\mathcal{A}$  cannot be a Boolean realizer of  $\mathcal{A}$  because it cannot complete all inputs in the domain with an output in the relation  $R(\mathcal{A})$ . We now show that we can compute in polynomial time for a given weighted specification  $\mathcal{A}$  a sub-automaton  $\mathcal{A}'$  of  $\mathcal{A}$  that is domain-safe and has the same Boolean realizers as  $\mathcal{A}$ . We would like to mention that there is a tight connection between domain-safe automata and the problem of “determinization by pruning” (DBP) as it is studied in [3]. The following result can also be derived from the proof of [3, Theorem 4.1]. Furthermore, the proof of Theorem 2 directly yields an alternative game-based proof of the “determinization by pruning” problem.

► **Theorem 2.** *There is a polynomial time procedure that takes as input a weighted specification  $\mathcal{A}$ , and either returns “no realizer” if  $\mathcal{A}$  does not have Boolean realizers, or, otherwise, returns a sub-automaton  $\mathcal{A}'$  of  $\mathcal{A}$  that is domain-safe, has the same domain as  $\mathcal{A}$ , and has the same Boolean realizers as  $\mathcal{A}$ .*

A direct consequence of the above theorem is that the Boolean synthesis problem is decidable in polynomial time.

### 3 Critical prefix games

In this section we introduce the necessary definitions and notations regarding games. Moreover, we introduce critical prefix games and establish our results for these kind of games.

**Games.** A *weighted game with imperfect information* is an infinite-duration two-player game played on a game arena  $G = (V, v_0, A, E, \mathcal{O}, w)$ , where  $V$  is a finite set of vertices,  $v_0 \in V$  is the initial vertex,  $A$  is a finite set of actions,  $E \subseteq V \times A \times V$  is a labeled transition relation,  $\mathcal{O} \subseteq 2^V$  is a set of *observations* that partition  $V$ , and  $w: E \rightarrow \mathbb{Z}$  is a weight function. Without loss of generality, we assume that the arena has no dead ends, i.e., for all  $v \in V$  there exists  $a \in A$  and  $v' \in V$  such that  $(v, a, v') \in E$ . The unique observation containing a vertex  $v$  is denoted  $\text{obs}(v)$ . A game with *perfect information* is such that  $\mathcal{O} = \{\{v\} \mid v \in V\}$ . In that case we omit  $\mathcal{O}$  from the tuple  $G$ .

Games are played in rounds in which Eve chooses an action  $a \in A$ , and Adam chooses an  $a$ -successor of the current vertex. The first round starts in the initial vertex  $v_0$ . A *play*  $\pi$  in  $G$  is an infinite sequence  $v_0 a_0 v_1 a_1 \dots$  such that  $(v_i, a_i, v_{i+1}) \in E$  for all  $i \in \mathbb{N}$ . The prefix of  $\pi$  up to  $v_n$  is denoted  $\pi(n)$ , its last element  $v_n$  is denoted by  $\text{last}(\pi(n))$ . The set of all plays resp. prefixes of plays in  $G$  is denoted by  $\text{Plays}(G)$  resp.  $\text{Prefs}(G)$ . The *observation sequence* of the play  $\pi$  is defined as  $\text{obs}(\pi) = \text{obs}(v_0)a_0\text{obs}(v_1)a_1\dots$  and the finite observation sequence of the play prefix  $\pi(n)$  is  $\text{obs}(\pi(n)) = \text{obs}(v_0)a_0\dots\text{obs}(v_n)$ . Naturally,  $\text{obs}$  extends to sets of (prefixes of) plays.

A game is defined by an arena  $G$  and an *objective*  $\text{Win} \subseteq \text{Plays}(G)$  describing a set of good plays in  $G$  for Eve. A *strategy for Eve* in  $G$  is a mapping  $\sigma: \text{Prefs}(G) \rightarrow A$ , it is called *observation-based* if for all play prefixes  $\rho, \rho' \in \text{Prefs}(G)$ , if  $\text{obs}(\rho) = \text{obs}(\rho')$ , then  $\sigma(\rho) = \sigma(\rho')$ . Equivalently, an observation-based strategy is a mapping  $\sigma: \text{obs}(\text{Prefs}(G)) \rightarrow A$ . We do not formally introduce strategies for Adam, intuitively, given a play prefix and an action  $a$ , a strategy of Adam selects an  $a$ -successor of its last vertex. Given a strategy  $\sigma$ , let  $\text{Plays}_\sigma(G)$  denote the set of plays compatible with  $\sigma$  in  $G$ , and  $\text{Prefs}_\sigma(G)$  denote the set of play prefixes of  $\text{Plays}_\sigma(G)$ . An Eve's strategy  $\sigma$  in  $G$  is *winning* if  $\text{Plays}_\sigma(G) \subseteq \text{Win}$ .

We now define quantitative objectives. The *energy level* of the play prefix  $\pi(n)$  is  $\text{EL}(\pi(n)) = \sum_{i=1}^n w((v_{i-1}, a_{i-1}, v_i))$ , the *sum value* is  $\text{Sum}(\pi(n)) = \sum_{i=1}^n w((v_{i-1}, a_{i-1}, v_i))$ , the *average value* is  $\text{Avg}(\pi(n)) = \frac{1}{n} \text{Sum}(\pi(n))$ , and the *discounted-sum value* is  $\text{Dsum}(\pi(n)) = \sum_{i=1}^n \lambda^i w((v_{i-1}, a_{i-1}, v_i))$ , and we let  $\text{Dsum}(\pi) = \sum_{i=1}^{\infty} \lambda^i w((v_{i-1}, a_{i-1}, v_i))$  (we do not explicitly mention the discount factor  $\lambda$  in this notation because it is always clear from the context).

The *energy objective* in  $G$  is parameterized by an initial credit  $c_0 \in \mathbb{N}$  and is given by  $\text{PosEn}_G(c_0) = \{\pi \in \text{Plays}(G) \mid \forall i \in \mathbb{N}: c_0 + \text{EL}(\pi(i)) \geq 0\}$ . It requires that the energy level of a play never drops below zero when starting with initial energy level  $c_0$ . The *fixed initial credit problem for imperfect information games* asks whether there exists an observation-based winning strategy for Eve for the objective  $\text{PosEn}_G(c_0)$ . The *discounted-sum objective* in  $G$  is parameterized by a threshold  $\nu \in \mathbb{Q}$ , and  $\triangleright \in \{>, \geq\}$ . It is given by  $\text{DS}_G^\triangleright(\nu) = \{\pi \in \text{Plays}(G) \mid \text{Dsum}(\pi) \triangleright \nu\}$  and requires that the discounted-sum value of a play is greater than resp. at least  $\nu$ . The *discounted-sum game problem* asks whether there exists a winning strategy for Eve for the objective  $\text{DS}_G^\triangleright(\nu)$ .

A game with perfect information is a special case of an imperfect information game. Classically, instead of using the above model with full observation, a (weighted) perfect information game, simply called game, is defined over an arena  $(V, V_\exists, v_0, E, w)$ , where the set of vertices  $V$  is partitioned into  $V_\exists$  and  $V \setminus V_\exists$ , the vertices belonging to Eve and Adam,

respectively,  $v_0 \in V$  is the initial vertex,  $E \subseteq V \times V$  is a transition relation, and  $w: E \rightarrow \mathbb{Z}$  is a weight function. In a play on such a game arena, Eve chooses a successor if the current vertex belongs to her, otherwise Adam chooses. For games with perfect information the two models are equivalent and we shall use both.

**Critical prefix games.** A *critical prefix game* is a game, where the winning objective is parameterized by a set  $C \subseteq V$  of *critical vertices*, and a set of play prefixes  $W \subseteq \text{Prefs}(G)$ . Its objective is defined as  $\text{Crit}_{C,W}(G) = \{\pi \in \text{Plays}(G) \mid \forall i \text{ last}(\pi(i)) \in C \rightarrow \pi(1) \dots \pi(i) \in W\}$ . The idea of a critical prefix game is that the state of a play is only relevant whenever the play is in a critical vertex. For convenience, in the case of critical prefix games, we also refer to the set  $W$  as objective.

The *threshold problem for critical prefix games* asks whether there exists a winning strategy for Eve for the objective  $\text{Crit}_{C,W}(G)$ , where  $W$  is of the form  $\text{Thres}_G^{\triangleright}(\nu) = \{\varphi \in \text{Prefs}(G) \mid V(\varphi) \triangleright \nu\}$  parameterized by a threshold  $\nu \in \mathbb{Q}$ ,  $\triangleright \in \{>, \geq\}$ , and  $V \in \{\text{Sum}, \text{Avg}, \text{Dsum}\}$ .

The *initial credit problem for critical prefix imperfect information energy games* asks whether there exists an observation-based winning strategy for Eve for the objective  $\text{Crit}_{C,W}(G)$ , where  $W$  is of the form  $\text{PrefPosEn}_G(c_0) = \{\varphi \in \text{Prefs}(G) \mid c_0 + \text{EL}(\varphi) \geq 0\}$  parameterized by an initial credit  $c_0 \in \mathbb{N}$ .

► **Theorem 3.** *The threshold problem for critical prefix games for  $V \in \{\text{Sum}, \text{Avg}\}$  and a strict or non-strict threshold is decidable in  $\text{NP} \cap \text{coNP}$ . Moreover, positional strategies are sufficient for Eve to win such games.*

**Proof sketch.** For Sum and Avg and a strict or non-strict threshold, the critical prefix threshold games reduce to mean-payoff games which are solvable in  $\text{NP} \cap \text{coNP}$  [34]. Positional strategies suffice for mean-payoff games, a winning strategy in the constructed mean-payoff game directly yields a positional winning strategy in the critical prefix threshold game. ◀

► **Theorem 4.** *The threshold problem for critical prefix games for Dsum and a strict resp. non-strict threshold is decidable in NP resp.  $\text{NP} \cap \text{coNP}$ . Moreover, positional strategies are sufficient for Eve to win such games.*

To prove the above theorem, we first show a result on weighted graphs which is interesting in itself.

► **Lemma 5.** *Given a weighted graph  $G$ , a source vertex  $v_0 \in V$ , a target set  $T \subseteq V$  and a threshold  $\nu \in \mathbb{Q}$ , checking whether there exists a path  $\pi$  from  $v_0$  to some vertex  $v \in T$  such that  $\text{Dsum}(\pi) \leq \nu$  can be done in PTIME.*

Lemma 5 can be used to show that the  $\geq \nu$ -non-emptiness problem for nondeterministic discounted-sum automata<sup>3</sup> can be checked in PTIME, a result which is, to the best of our knowledge, new. It was known to be in PSPACE for unambiguous discounted-sum automata [25, 9]. This problem asks for the existence of a word of value greater or equal than a given threshold  $\nu$ . Since the value of a word is the maximal value amongst its accepting runs, it suffices to check for the existence of a run from the initial state to an accepting state of discounted-sum value  $\geq \nu$ . By inverting the weights, the latter is equivalent to checking

<sup>3</sup> In contrast to deterministic weighted automata, there might be several accepting runs on an input and the value of the word is defined as the maximal value of its accepting runs [25, 28].

whether there exists a run from the initial state to an accepting state of discounted-sum value  $\leq -\nu$ . By seeing the (inverted) discounted-sum automaton as a weighted graph, the latter property can be checked in PTIME by Lemma 5, thus proving the following theorem.

► **Theorem 6.** *The  $\geq \nu$  non-emptiness problem is decidable in PTIME for nondeterministic discounted-sum automata.*

We now go back to the proof of Theorem 4.

**Proof sketch of Theorem 4.** For Dsum, and a non-strict threshold, the problem can be directly reduced to discounted-sum games which are solvable in  $\text{NP} \cap \text{CONP}$  [4].

For Dsum, and a strict threshold, such a reduction fails. To solve the problem, we first show that positional strategies are sufficient for Eve to win in a critical prefix threshold discounted-sum game (for strict and non-strict thresholds). The NP-algorithm guesses a positional strategy  $\sigma$  for Eve, and then verifies in polynomial time whether  $\sigma$  is winning. Let  $G'$  be the game restricted to Eve's  $\sigma$ -edges, seen as a weighted graph. The strategy  $\sigma$  is not winning iff Adam can form a path in  $G'$  from the initial vertex to a critical vertex that has weight  $\leq \nu$ . This property can be checked in PTIME thanks to Lemma 5 (by taking as target set the set of critical vertices). ◀

The following is shown by reduction from the halting problem for 2-counter machines.

► **Theorem 7.** *The fixed initial credit problem for imperfect information critical prefix energy games is undecidable.*

The above result contrasts the fixed initial credit problem for imperfect information energy games which is decidable [22].

► **Theorem 8.** *The fixed initial credit problem for imperfect information critical prefix energy games is decidable if from each vertex Adam has a strategy to reach a critical vertex against observation based strategies. Moreover, finite-memory strategies are sufficient for Eve to win.*

**Proof sketch.** This problem is reduced to the fixed initial credit problem for imperfect information energy games which is decidable [22]. In classical energy games, Eve loses as soon as the energy goes below zero. The idea of the reduction is that if in the critical prefix energy game the initial credit is  $c_0$ , then in the classical energy game we start the game with an additional buffer, i.e., with  $c_0 + B$ , for some computable bound  $B$ . In the critical prefix energy game, if the energy level drops below  $-B$  Adam can force to visit a critical vertex such that the energy level can rise by at most  $B$ , ensuring that a critical vertex is visited with energy level below zero. Thus, the additional buffer  $B$  suffices in the classical energy game. ◀

## 4 Synthesis problems

Here, we solve the quantitative synthesis problems defined in Section 2. Recall that weighted specifications are given by weighted automata that alternate between reading one input and one output symbol. In other words, we prove the decidability results of Table 1. We then show consequences of these results to quantitative synthesis problems over infinite words.

**Threshold synthesis problems.** Since weighted specifications  $S$  are given by weighted automata, the synthesis problem naturally reduces to a game played on the automaton. In order to solve threshold synthesis problems, in contrast to best-value and approximate synthesis problems, it is not necessary to compare the values of runs of the specification automaton that have the same input sequence. Hence, it is relatively straightforward to reduce threshold synthesis problems to critical prefix threshold games. An important point needs to be taken care of due to the fact the domain of  $S$  might be partial, and therefore lead Eve into the following bad situation ( $\star$ ): Eve must choose her outputs in such a way that she does not go in a state of the automaton which is non-accepting, while the input word played by Adam so far is in the domain of  $S$ . Otherwise, the pair of input and output word formed would not even be in  $S$ , something which is required by the definition of synthesis problems. So, Eve has to monitor the domain, which is easy if the domain is total, but more involved if it is partial. Thanks to Theorem 2, this can be done in polynomial time. More precisely, we first run the algorithm of Theorem 2 which either returns that there is no Boolean realizer, or returns a domain-safe deterministic weighted automaton  $\mathcal{A}'$  which has the same Boolean realizers as  $S$ . By the very definition of domain-safe automata, the bad situation ( $\star$ ) described above cannot happen. Hence, Eve can freely play on  $\mathcal{A}'$  without taking care of the domain constraint. Only the quantitative constraint matters, and it has to be enforced whenever Eve is in an accepting state of  $\mathcal{A}'$  (this corresponds to the situation where Adam has chosen an input word in the domain of  $S$ ). Hence, only accepting states of  $\mathcal{A}'$  matter for the quantitative constraint and these are declared as critical. To conclude, by projecting away the symbols of  $\mathcal{A}'$  and by declaring its accepting states to be critical, we obtain a critical prefix game. For the threshold synthesis problem, decidability follows directly from the decidability of the threshold problem for critical prefix games (Theorems 3 and 4). For Sum- and Avg-specifications, this can be done in  $\text{NP} \cap \text{coNP}$ . We leave open whether it is solvable in PTIME and show that this would also solve the long standing open problem of whether mean-payoff games are solvable in PTIME.

► **Theorem 9.** *The threshold synthesis problem for a  $V$ -specification with  $V \in \{\text{Sum}, \text{Avg}\}$  and a strict or non-strict threshold is decidable in  $\text{NP} \cap \text{coNP}$  and PTIME-equivalent to mean-payoff games. The threshold synthesis problem for a Dsum-specification and a strict resp. non-strict threshold is decidable in NP resp. in  $\text{NP} \cap \text{coNP}$ .*

**Synthesis and regret determinization.** Before we prove our results about best-value and approximate synthesis, we highlight the tight connection between the approximate synthesis problem and the so-called regret determinization problem for nondeterministic weighted automata<sup>4</sup>. This problem has for instance been studied in [26] for Sum-automata and in [29] for Dsum-automata. We formalize this connection here. Given  $r \in \mathbb{Q}$  and  $\triangleleft \in \{<, \leq\}$ , a nondeterministic WFA  $\mathcal{A} = (Q, \Sigma, q_i, \Delta, F, \gamma)$  is called  $r_{\triangleleft}$ -regret determinizable if there exists a finite set of memory states  $M$  and a deterministic WFA  $\mathcal{A}_r = (Q \times M, \Sigma, q_i^r, \Delta_r, F_r, \gamma_r)$ , where  $q_i^r = (q_i, m)$  for some  $m \in M$ ,  $F_r \subseteq F \times M$ ,  $((q, m), a, (q, m')) \in \Delta_r$  implies that  $(q, a, q') \in \Delta$ , and  $\gamma_r(((q, m), a, (q, m')))) = \gamma((q, a, q'))$  for all  $m, m' \in M$ , such that  $L(\mathcal{A}) = L(\mathcal{A}_r)$  and  $\mathcal{A}(w) - \mathcal{A}_r(w) \triangleleft r$  for all  $w \in \text{dom}(L(\mathcal{A}))$ . The *regret determinization problem* asks, given a nondeterministic weighted automaton  $\mathcal{A}$ , a threshold  $r \in \mathbb{Q}$ , and  $\triangleleft \in \{<, \leq\}$ , whether  $\mathcal{A}$  is  $r_{\triangleleft}$ -regret determinizable.

<sup>4</sup> In contrast to deterministic weighted automata, there might be several accepting runs on an input and the value of the word is defined as the maximal value of its accepting runs [25, 28].

► **Lemma 10.** *The approx. synthesis problem for weighted specifications reduces in linear time to the regret determinization problem for nondet. weighted automata (with the same threshold). The converse is true (in linear time and with the same threshold) for Sum-automata.*

Lemma 10 is independent from any payoff function. Regarding the converse direction, when going from the regret determinization problem to the approximate synthesis problem, a transition (for an input symbol) must be translated into two transitions (adding an output symbol). This step can cause difficulties depending on the used payoff function, e.g., Dsum.

**Best-value synthesis problems.** Best-value synthesis is equivalent to zero-regret synthesis, which is, by Lemma 10, equivalent to zero-regret determinization of weighted automata. In [9], the authors showed that if a Sum-automaton is zero-regret determinizable, then no memory states are needed, i.e., a sub-automaton suffices. We give general sufficient conditions on weighted finite automata (which hold for Sum-, Avg- and Dsum-automata) under which the latter result can be generalized.

Let  $V: \mathbb{Z}^* \rightarrow \mathbb{Q}$  be a payoff function. A  $V$ -automaton defining a  $V$ -specification, where  $V$  is applied to runs as usual, is called  $\leq$ -stable if for all runs  $\rho, \rho', \rho''$  such that the end state of  $\rho$  is the beginning state of  $\rho'$  and  $\rho''$ ,  $w' = u \otimes v'$ , and  $w'' = u \otimes v''$  for some  $u \in \Sigma_a^*$  and  $v', v'' \in \Sigma_o^*$ , where  $w'$  and  $w''$  are the words associated to  $\rho'$  and  $\rho''$ , respectively, holds that if  $V(\rho') \leq V(\rho'')$  then  $V(\rho\rho') \leq V(\rho\rho'')$ .

► **Lemma 11.** *Given a weighted specification  $S$  by a  $\leq$ -stable weighted automaton  $\mathcal{A}$ , if there exists a transducer that implements a best-value  $S$ -realization, then there exists a transducer that implements a best-value  $S$ -realization that is defined as a sub-automaton of  $\mathcal{A}$ .*

While the above lemma can be used to obtain our decidability results for best-value synthesis, we use other techniques to obtain the complexity results stated below.

► **Theorem 12.** *The best-value synthesis problem is decidable in PTIME for Sum-specifications and Avg-specifications, and in  $NP \cap \text{coNP}$  for Dsum-specifications.*

**Proof sketch.** For Sum, the problem reduces to the zero-regret determinization problem for Sum-automata, see Lemma 10, aka the determinization by pruning problem for Sum-automata, known to be decidable in PTIME in [3]. For Avg, it easily reduces to Sum by interpreting the Avg-specification as a Sum-specification. For Dsum, we show that the problem reduces in PTIME to a critical prefix threshold game, for non-strict threshold, which is solvable in  $NP \cap \text{coNP}$  by Theorem 4. ◀

Alternatively, decidability for Dsum can be obtained by reduction to the zero-regret determinization problem for Dsum-automata over infinite words which was shown to be decidable in NP in [29, Theorem 6]. However, our techniques allow us to get  $NP \cap \text{coNP}$ .

**Approximate synthesis problems.** We now turn to the approximate synthesis problems and show its decidability for Sum and Avg. We leave the decidability status open for Dsum, but nevertheless show decidability for a large class, namely when the discount factor is of the form  $\frac{1}{n}$  for  $n \in \mathbb{N}$ . Nondeterministic Dsum-automata in this class have been considered in [8] and shown to be determinizable.

► **Theorem 13.** *The approximate synthesis problem is*

- *EXPTIME-complete for Sum-specifications and strict or non-strict thresholds;*
- *decidable and EXPTIME-hard for Avg-specifications and strict or non-strict thresholds;*
- *in NEXPTIME (resp. EXPTIME) for Dsum-specifications with a discount factor  $\lambda$  of the form  $\frac{1}{n}$  with  $n \in \mathbb{N}$  and strict (resp. non-strict) thresholds.*

**Proof sketch.** For **Sum**, we reduce the problem to  $r$ -regret determinization of **Sum**-automata, known to be EXPTIME-complete, using the back-and-forth connection given by Lemma 10.

For an **Avg**-specifications  $S$ , it is worth noting that even though  $r$ -approximate synthesis reduces to  $r$ -approximate synthesis for **Sum** when  $r = 0$ , interpreting  $S$  as a **Sum**-specification, this reduction is wrong for  $r > 0$  in general. It is because in an **Avg**-specification, Eve can deviate more and more from the best sum, while the average of this difference can stay low. We instead rely on a reduction to critical prefix energy games of imperfect information and fixed initial credit (which falls into the decidable subclass of Theorem 8). Intuitively, in this game, Adam constructs a run  $\rho$  on a pair of words  $(u, v)$  and Eve constructs a run  $\rho'$  on some  $(u, v')$ . She only sees  $u$  and not  $\rho$ . The energy level of such a play is set to  $\text{Sum}(\rho') + |uv| \cdot r - \text{Sum}(\rho)$  and must be positive whenever Adam reaches an accepting state. EXPTIME-hardness is perhaps the most technical result of the paper, and is a non-trivial adaptation of reduction from countdown games used to show EXPTIME-hardness of the regret determinization of **Sum**-automata [26].

Finally, for **Dsum**, we use that by projecting away the output in the **Dsum**-automaton defining the specification, we obtain a nondeterministic weighted automaton which is determinizable by [8]. This allows us to reduce the problem to the threshold synthesis problem for **Dsum**, which is decidable by Theorem 9. To obtain the complexity results, we first analyze the determinization procedure. It yields an automaton whose states are exponential in the number of states and polynomial in the weights of the nondeterministic one. Its weights are polynomial in the weights of the nondeterministic one. For a strict threshold, the claimed complexity bound follows directly from Theorem 9. For a non-strict threshold, we use that critical prefix threshold games are reduced in polynomial time to discounted-sum games. Using value iteration [34] to solve discounted-sum games yields the claimed complexity bound, because it runs in polynomial time in the size of the arena, logarithmic in the absolute maximal weight of the arena, and exponential in the representation of the discount factor, i.e., polynomial in the discount factor. ◀

**Infinite words and Church synthesis.** An  $\omega$ -specification is a subset  $S \subseteq (\Sigma_{\mathfrak{i}}, \Sigma_{\mathfrak{o}})^\omega$ . The (Church) synthesis problem asks to decide whether there exists a strategy to pick a correct output sequence given longer and longer prefixes of an infinite input sequence. Formally, an  $\omega$ -specification  $S$  is said to be *realizable* if there exists a function  $\lambda: \Sigma_{\mathfrak{i}}^* \rightarrow \Sigma_{\mathfrak{o}}$  such that for all  $i_1 i_2 \dots \in \Sigma_{\mathfrak{i}}^\omega$ , it holds that  $i_1 \lambda(i_1) i_2 \lambda(i_1 i_2) i_3 \lambda(i_1 i_2 i_3) \dots \in S$ .

Strategies of interest are those which can be represented by a finite-state machine, and in particular a Mealy machine, that is, roughly, a transducer running on  $\omega$ -words and without acceptance condition. Formally, it is a tuple  $M = (P, p_0, \delta)$  such that  $P$  is a finite set of states with initial state  $p_0$ , and  $\delta: P \times \Sigma_{\mathfrak{i}} \rightarrow \Sigma_{\mathfrak{o}} \times P$  is a (total) transition function. The function  $\delta$  can be extended to  $\delta^*: P \times \Sigma_{\mathfrak{i}}^+ \rightarrow \Sigma_{\mathfrak{o}} \times P$  as usual. Then,  $M$  defines the strategy  $\lambda_M$  such that for all  $u \in \Sigma_{\mathfrak{i}}^*$ ,  $\lambda_M(u) = \pi_1(\delta^*(p_0, u))$ , where  $\pi_1$  is the first projection. It is well-known that when  $S$  is  $\omega$ -regular (given e.g. as a parity automaton), it is decidable whether  $S$  is realizable [14]. Moreover, realizability implies realizability by a Mealy machine.

**Weighted safety specifications.** In this paper, we go beyond  $\omega$ -regular specifications, by considering safety  $\omega$ -specifications induced by weighted specifications of finite words defined by deterministic weighted automata. Let  $W: (\Sigma_{\mathfrak{i}} \Sigma_{\mathfrak{o}})^* \rightarrow \mathbb{Q} \cup \{-\infty\}$  be a weighted specification. For a threshold  $t \in \mathbb{Q}$  and  $\triangleright \in \{>, \geq\}$ , we define the  $\omega$ -specification  $\text{Thres}^{\triangleright t}(W) = \{i_1 o_1 \dots \in (\Sigma_{\mathfrak{i}}, \Sigma_{\mathfrak{o}})^\omega \mid \forall k \geq 0, i_1 \dots i_k \in \text{dom}(W) \rightarrow W(i_1 o_1 \dots i_k o_k) \triangleright t\}$ . In words, an  $\omega$ -word  $w$  is in  $\text{Thres}^{\triangleright t}(W)$  iff for all finite prefixes  $u = i_1 o_1 \dots i_k o_k$  of  $w$ , either  $i_1 \dots i_k \notin \text{dom}(W)$  or

$W(u) \triangleright t$ . So, the quantitative condition is checked only for prefixes whose input belongs to  $\text{dom}(W)$ . The  $\omega$ -specification  $\text{Thres}^{\triangleright t}(W)$  is a safety specification<sup>5</sup>. More generally, any set  $S \subseteq (\Sigma_{\mathfrak{i}}.\Sigma_{\mathfrak{o}})^*$  induces a safety  $\omega$ -specification  $\text{Safe}(S) = \{i_1 o_1 \dots \in (\Sigma_{\mathfrak{i}}.\Sigma_{\mathfrak{o}})^\omega \mid \forall k \geq 0, i_1 \dots i_k \in \text{dom}(S) \rightarrow i_1 o_1 \dots i_k o_k \in S\}$ .

For example, we have the equality  $\text{Thres}^{\triangleright t}(W) = \text{Safe}(\{u \in (\Sigma_{\mathfrak{i}}.\Sigma_{\mathfrak{o}})^* \mid W(u) \triangleright t\})$ . Likewise, we define best-value and approximate safety  $\omega$ -specifications. Formally, given a finite word  $i_1 \dots i_k \in \Sigma_{\mathfrak{i}}^*$  and  $\triangleleft \in \{<, \leq\}$ , we let  $\text{BestVal}(W) = \text{Approx}^{\leq}(W, 0)$  where for all  $r \in \mathbb{Q}_{\geq 0}$  we have  $\text{Approx}^{\triangleleft}(W, r) = \text{Safe}(\{u = i_1 o_1 \dots i_k o_k \mid \text{bestVal}_W(i_1 \dots i_k) - W(u) \triangleleft r\})$ . Note that the three notions of safety  $\omega$ -specifications we have defined are not necessarily  $\omega$ -regular, even if  $W$  is given by a deterministic weighted automaton. Nevertheless, an immediate consequence of the results we have obtained previously on finite words is that

► **Theorem 14.** *The synthesis problem for an  $\omega$ -specification  $O \subseteq (\Sigma_{\mathfrak{i}}.\Sigma_{\mathfrak{o}})^\omega$  is decidable when  $O$  is given by a deterministic  $V$ -automaton defining a weighted  $V$ -specification of finite words  $W$  s.t.  $O \in \{\text{Thres}^{\triangleright t}(W), \text{Thres}^{\geq t}(W), \text{BestVal}(W), \text{Approx}^<(W, r), \text{Approx}^{\leq}(W, r)\}$  and  $V = \text{Sum}$ ,  $V = \text{Avg}$  or  $V = \text{Dsum}$  with discount factor  $1/n$  for  $n \in \mathbb{N}$ . Moreover, if  $O$  is realizable, it is realizable by a Mealy machine.*

## 5 Future work

In this paper, weighted specifications are defined by deterministic weighted automata. Nondeterministic, even unambiguous, weighted automata, are strictly more expressive than their deterministic variant in general, and in particular for  $\text{Sum}$ ,  $\text{Avg}$  and  $\text{Dsum}$ . An interesting direction is to revisit our quantitative synthesis problems for specifications defined by nondeterministic weighted automata. Using similar ideas as the undecidability of critical prefix energy games of imperfect information, it can be shown that threshold synthesis becomes undecidable for unambiguous sum- and avg-specifications. The problem is open for best-value and approximate synthesis, and we plan to investigate it.

Two other directions seem interesting as future work, both in the setting of infinite words. First, natural measures in this setting are discounted-sum and mean-payoff. While the threshold synthesis problems directly reduce to known results and best-value/approximate synthesis for  $\text{dsum}$  has been studied in [29], nothing is known to the best of our knowledge about best-value/approximate synthesis for mean-payoff. We expect the techniques to be different because such a measure is prefix-independent, unlike our measures in the setting of finite words. As a second direction, we have seen how our results apply to synthesis on infinite words through weighted safety conditions. An interesting direction is to consider such weighted requirements in conjunction with  $\omega$ -regular conditions such as parity, in the line of [17] that combines energy and parity objectives in games.

---

## References

- 1 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *International Conference on Computer Aided Verification*, pages 541–563. Springer, 2020.
- 2 Shaull Almagor, Orna Kupferman, Jan Oliver Ringert, and Yaron Vener. Quantitative assume guarantee synthesis. In *International Conference on Computer Aided Verification*, pages 353–374. Springer, 2017.

---

<sup>5</sup> A language of  $\omega$ -words  $S$  is a safety language if any  $\omega$ -word  $w$  whose finite prefixes  $u$  are such that  $uv_u \in S$  for some  $\omega$ -word  $v_u$ , belongs to  $S$  [15].

- 3 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.
- 4 Daniel Andersson. An improved algorithm for discounted payoff games. In *ESSLLI Student Session*, pages 91–98, 2006.
- 5 Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *International Conference on Computer Aided Verification*, pages 140–156. Springer, 2009.
- 6 Roderick Bloem, Rüdiger Ehlers, and Robert Könighofer. Cooperative reactive synthesis. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, volume 9364 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2015.
- 7 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. doi:10.1016/j.jcss.2011.08.007.
- 8 Udi Boker and Thomas A. Henzinger. Exact and approximate determinization of discounted-sum automata. *Logical Methods in Computer Science*, 10(1), 2014.
- 9 Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 750–761. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.74.
- 10 Patricia Bouyer, Uli Fahrenberg, Kim G Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 61–70, 2010.
- 11 Tomáš Brázdil, Petr Jančar, and Antonín Kučera. Reachability games on extended vector addition systems with states. In *International Colloquium on Automata, Languages, and Programming*, pages 478–489. Springer, 2010.
- 12 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications*, pages 3–23. Springer, 2016.
- 13 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017. doi:10.1007/s00236-016-0273-2.
- 14 J Richard Büchi and Lawrence H Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.
- 15 Edward Chang, Zohar Manna, and Amir Pnueli. The safety-progress classification. In *Logic and Algebra of Specification*, pages 143–202. Springer, 1993.
- 16 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theoretical Computer Science*, 458, 2012.
- 17 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.
- 18 Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2007.
- 19 A Church. Applications of recursive arithmetic to the problem of circuit synthesis—summaries of talks. *Institute for Symbolic Logic, Cornell University*, 1957.
- 20 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018.
- 21 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages,*

- and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 22 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In *International Workshop on Computer Science Logic*, pages 260–274. Springer, 2010.
  - 23 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.
  - 24 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.
  - 25 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2012.
  - 26 Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *LICS*, pages 1–12. IEEE Computer Society, 2017.
  - 27 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.
  - 28 Axel Haddad and Benjamin Monmege. Why value iteration runs in pseudo-polynomial time for discounted-payoff games. *Technical note, Université libre de Bruxelles*, 2015.
  - 29 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Minimizing regret in discounted-sum games. In *CSL*, volume 62 of *LIPICs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
  - 30 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.
  - 31 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safrless compositional synthesis. In *Computer Aided Verification, 18th International Conference, CAV 2006*, volume 4144 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2006.
  - 32 Jianwen Li, Kristin Y. Rozier, Geguang Pu, Yueling Zhang, and Moshe Y. Vardi. Sat-based explicit ltl satisfiability checking. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 2946–2953. AAAI Press, 2019.
  - 33 Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic ltl synthesis. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1362–1369. ijcai.org, 2017.
  - 34 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

# Reachability for Updatable Timed Automata Made Faster and More Effective

Paul Gastin 

LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France  
paul.gastin@ens-paris-saclay.fr

Sayan Mukherjee 

Chennai Mathematical Institute, India  
sayanm@cmi.ac.in

B Srivathsan 

Chennai Mathematical Institute, India  
sri@cmi.ac.in

---

## Abstract

---

Updatable timed automata (UTA) are extensions of classical timed automata that allow special updates to clock variables, like  $x := x - 1$ ,  $x := y + 2$ , etc., on transitions. Reachability for UTA is undecidable in general. Various subclasses with decidable reachability have been studied. A generic approach to UTA reachability consists of two phases: first, a static analysis of the automaton is performed to compute a set of clock constraints at each state; in the second phase, reachable sets of configurations, called zones, are enumerated. In this work, we improve the algorithm for the static analysis. Compared to the existing algorithm, our method computes smaller sets of constraints and guarantees termination for more UTA, making reachability faster and more effective. As the main application, we get an alternate proof of decidability and a more efficient algorithm for timed automata with bounded subtraction, a class of UTA widely used for modelling scheduling problems. We have implemented our procedure in the tool TChecker and conducted experiments that validate the benefits of our approach.

**2012 ACM Subject Classification** Theory of computation → Timed and hybrid models

**Keywords and phrases** Updatable timed automata, Reachability, Zones, Simulations, Static analysis

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.47

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2009.13260>.

**Funding** Work supported by IRL ReLaX. Paul Gastin is supported by ANR project TickTac (ANR-18-CE40-0015). B Srivathsan is supported by CEFIPRA project IoTTTA (DST/CNRS ref. 2016-01). Sayan Mukherjee and B Srivathsan are additionally supported by Infosys Foundation (India).

## 1 Introduction

Timed automata [1] are finite automata equipped with real-time variables called clocks. Values of the clock variables increase at the same rate as time progresses. Transitions are guarded by constraints over the clock variables. During a transition, the value of a variable can be updated in several ways. In the classical model, variables can be reset to 0, written as a command  $x := 0$  in transitions. Generalizations of this involve  $x := c$  with  $c \geq 0$  or  $x := y + d$  where  $d$  is an arbitrary integer. Automata with these more general updates are called *Updatable Timed Automata (UTA)* [8, 6]. The updates provide a “discrete jump” facility during transitions. These are useful syntactic constructs for modeling real-time systems and have been used in several studies [12, 23, 19, 18, 25].

On the one hand, variables with both a continuous and a discrete flow offer modeling convenience. On the other hand, the discrete jumps are powerful enough to simulate counter machines through the use of  $x := x + 1$  and  $x := x - 1$  updates, in fact with zero time



© Paul Gastin, Sayan Mukherjee, and B Srivathsan;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 47; pp. 47:1–47:17



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

elapse during the entire simulation [8]. This makes reachability for this model undecidable. Various decidable subclasses have been investigated over the years [8, 12]. The most common technique to prove decidability involves showing the existence of a region automaton [1], which is a finite automaton accepting the (untimed) sequences of actions that have a timed run in the UTA. Although this gives decidability, the algorithm via the region construction is impractical due to the presence of exponentially many regions. Practical algorithms in current tools like UPPAAL [24], PAT [27], Theta [28] and TChecker [20] work with *zones*, which are bigger sets of configurations than regions and can be efficiently represented and manipulated using Difference-Bound Matrices (DBMs) [10]. Notably, these tools implement zone based algorithms only for UTA with restricted updates  $x := c$  for  $c \geq 0$ , which behave similar to the reset  $x := 0$ . Most of the efforts in making the zone based algorithm more efficient have concentrated on this subclass of timed automata with only resets [4, 22, 26].

Recently, we have presented a zone based algorithm for updatable timed automata [14]. Due to the undecidability of the problem, it cannot deal with the whole class of UTA. It however covers the subclasses tabulated in [8]. The algorithm consists of two phases: first, a static analysis of the automaton is performed to compute a set of clock constraints at each state of the automaton; in the second phase, reachable sets of configurations, stored as zones, are enumerated. None of these phases has a guaranteed termination. If the static analysis terminates, a simulation relation between zones based on the constraints generated in the static analysis can be used to guarantee termination of the zone enumeration. Moreover, a smaller set of constraints in the static analysis gives a coarser simulation which leads to a faster zone enumeration. The simulation used in [14] lifts the efficient *LU*-simulation [4, 22] studied for diagonal-free reset-only timed automata to automata with diagonal constraints and updates.

**Contributions.** In this work, we strongly improve the static analysis of [14]. The new approach accumulates fewer clock constraints and terminates for a wider class of UTA. In particular, it terminates for *timed automata with bounded subtraction*, which was not the case before. This class contains updates  $x := x - c$  with  $c \geq 0$  along with resets. However, an update  $x := x - c$  is allowed in a transition only when there is a promise that each configuration that can take this transition has a bounded  $x$ -value. This boundedness property gives decidability thanks to a finite region equivalence. This class has been used to model schedulability problems [12], where updates  $x := x - c$  have been crucially used to model preemption. Thus, our new static analysis allows to use efficient simulations during the zone enumeration for this class.

At an algorithmic level, the new analysis is a slight modification of the older one. However, this makes some of the technical questions significantly harder: we show that deciding termination of the new analysis can be done in polynomial-time if the constants in the guards and updates are encoded in unary, whereas the problem is PSPACE-complete when the constants are encoded in binary. The older analysis does not depend on the encoding of constants, and has a polynomial-time algorithm for deciding termination.

For the experiments, the differences in the encoding and the hardness result do not carry much importance. The static analysis is implemented as a fixed-point iteration which can continue for a fixed number of steps determined by the size of the automaton, or can be stopped after a fixed time-out. We have implemented the new static analysis in the open source tool TChecker [20]. We noticed that the new method terminates and produces a result for more cases, and when both methods produce a result, the new method is faster.

**Related work.** Static analysis for timed automata without diagonal constraints and with updates restricted to  $x := c$  and  $x := y + c$  with  $c \geq 0$  was studied in [3] in the context of  $M$ -simulations, which were implemented in earlier versions of UPPAAL and KRONOS [29]. Latest tools implement a more efficient  $LU$ -simulation [4, 22]. Our method clarifies how some optimizations of [3] can be lifted to the context of  $LU$ -simulations and more general updates, and also provides additional optimizations. TIMES [2] is a tool for modeling scheduling problems. It is mentioned in [12] that TIMES implements an algorithm using zones based on “the UPPAAL DBM library extended with a subtraction operator”. However, the exact simulations used in the zone enumeration are not clear to us. A different approach to reachability is presented in [21] where the constraints needed for simulation are learnt during the zone enumeration directly. This potentially gives more relevant constraints and hence coarser simulations. On the flip side, it requires a sophisticated zone enumeration method with observable overheads. Moreover [21] deals with timed automata without diagonal constraints and general updates. Static analysis is lucrative since it is cheap, and maintains the reachability procedure as two simple steps. Apart from verification of UTA, studies on the expressive power of updates and diagonal constraints have been carried out in [8, 7]. Timed register automata [5] are a variant of UTA that have been looked at in the context of canonical representations.

**Organization.** Section 2 gives the preliminary definitions. Section 3 introduces the new static analysis approach. Some classes of UTA where the new static analysis performs better are discussed in Section 4. The subsequent Section 5 discusses the termination problem for the new static analysis. Section 6 provides the results of our experiments. We conclude with Section 7. The extended version [15] contains missing proofs and details about the models used for the experiments.

## 2 Preliminaries

We denote by  $\mathbb{R}$  the set of reals, by  $\mathbb{R}_{\geq 0}$  the non-negative reals, by  $\mathbb{Z}$  the integers and by  $\mathbb{N}$  the natural numbers. Let  $X$  be a finite set of variables over  $\mathbb{R}_{\geq 0}$  called *clocks*. A *valuation* is a function  $v: X \rightarrow \mathbb{R}_{\geq 0}$  that maps every clock to a non-negative real number. For  $\delta \in \mathbb{R}_{\geq 0}$  we define valuation  $v + \delta$  as  $(v + \delta)(x) := v(x) + \delta$ . The set of valuations is denoted by  $\mathbb{V}$ .

A *non-diagonal constraint* is an expression of the form  $x \triangleleft c$  or  $c \triangleleft x$ , where  $x \in X$ ,  $c \in \mathbb{N}$  and  $\triangleleft \in \{<, \leq\}$ , that is,  $x \triangleleft 3$  stands for either  $x < 3$  or  $x \leq 3$ . A *diagonal constraint* is an expression of the form  $x - y \triangleleft c$  or  $c \triangleleft x - y$  where  $x, y \in X$  are clocks and  $c \in \mathbb{N}$ . An *atomic constraint* is either a non-diagonal constraint or a diagonal constraint. We also consider two special atomic constraints  $\top$  (true) and  $\perp$  (false). A *constraint*  $\varphi$  is either an atomic constraint or a conjunction of atomic constraints, generated by the following grammar:  $\varphi ::= \top \mid \perp \mid x \triangleleft c \mid c \triangleleft x \mid x - y \triangleleft c \mid c \triangleleft x - y \mid \varphi \wedge \varphi$  with  $c \in \mathbb{N}$ ,  $\triangleleft \in \{<, \leq\}$ . Given a constraint  $\varphi$  and a valuation  $v$ , we write  $v(\varphi)$  for the boolean expression that we get by replacing every clock  $x$  present in  $\varphi$  with the value  $v(x)$ . A valuation  $v$  is said to *satisfy* a constraint  $\varphi$ , written as  $v \models \varphi$ , if the expression  $v(\varphi)$  evaluates to true. For every valuation  $v$ , we have  $v \models \top$  and  $v \not\models \perp$ . Given a constraint  $\varphi$ , we define the set  $\llbracket \varphi \rrbracket := \{v \mid v \models \varphi\}$ .

An *update*  $up: \mathbb{V} \mapsto \mathbb{V}$  is a partial function mapping valuations to valuations. The update  $up$  is specified by an *atomic update* for each clock  $x$ , given as either  $x := c$  or  $x := y + d$  where  $c \in \mathbb{N}$ ,  $d \in \mathbb{Z}$  and  $y \in X$  (is possibly equal to  $x$ ). We write  $up_x$  for the right hand side of the atomic update of  $x$ , that is,  $up_x$  is either  $c$  or  $y + d$ . Note that we want  $d$  to be an integer, since we allow for decrementing clocks, and on the other hand  $c \in \mathbb{N}$  since clock values are always non-negative. Given a valuation  $v$  and an update  $up$ , we define  $v(up_x)$  to

be  $c$  or  $v(y) + d$  depending on  $up_x$  being  $c$  or  $y + d$ . We say  $up(v) \geq 0$  if  $v(up_x) \geq 0$  for all  $x \in X$ . In this case the valuation  $up(v) \in \mathbb{V}$  is defined by  $up(v)(x) = v(up_x)$  for all  $x \in X$ . In general, due to the presence of updates  $up_x := y + d$  with  $d < 0$ , the update may not yield a clock valuation and for those valuations  $v$ ,  $up(v)$  is not defined. For example, if  $v(x) = 5$  and  $up_x = x - 10$  then  $up(v)$  is undefined. Hence, the domain of the partial function  $up: \mathbb{V} \rightarrow \mathbb{V}$  is the set of valuations  $v$  such that  $up(v) \geq 0$ . Updates can be used as transformations in timed automata transitions. An updatable timed automaton is an extension of a classical timed automaton which allows updates of clocks on transitions.

► **Definition 1.** An updatable timed automaton (UTA)  $\mathcal{A} = (Q, X, q_0, T, F)$  is given by a finite set  $Q$  of states, a finite set  $X$  of clocks, an initial state  $q_0$ , a set  $T$  of transitions and  $F \subseteq Q$  of accepting states. Transitions are of the form  $(q, g, up, q')$  where  $g$  is a constraint (also called guard) and  $up$  is an update,  $q, q' \in Q$  are the source and target states respectively.

Fix a UTA  $\mathcal{A} := (Q, X, q_0, T, F)$  for the rest of this section. A configuration of  $\mathcal{A}$  is a pair  $(q, v)$  with  $q \in Q$  and  $v \in \mathbb{V}$ . Semantics of  $\mathcal{A}$  is given by a transition system over its configurations. There are two kinds of transitions: *delay* and *action*. For every configuration  $(q, v)$  and every  $\delta \in \mathbb{R}_{\geq 0}$  there is a delay transition  $(q, v) \xrightarrow{\delta} (q, v + \delta)$ . For every transition  $t := (q, g, up, q')$  in the automaton, there is an action transition  $(q, v) \xrightarrow{t} (q', v')$  in the semantics if  $v \models g$  ( $v$  satisfies the guard),  $up(v) \geq 0$  (the update on  $v$  is defined) and  $v' = up(v)$ . The initial configuration is  $(q_0, v_0)$  with  $v_0(x) = 0$  for every clock  $x$ . We write  $(q, v) \xrightarrow{\delta, t} (q', v')$  for the sequence of delay  $\delta$  and action  $t$  from  $(q, v)$ . A *run* is an alternating sequence of delay and action transitions starting from the initial configuration:  $(q_0, v_0) \xrightarrow{\delta_1, t_1} (q_1, v_1) \xrightarrow{\delta_2, t_2} \dots \xrightarrow{\delta_n, t_n} (q_n, v_n)$ . The run is accepting if  $q_n \in F$ .

The reachability problem for UTA asks if a given UTA has an accepting run. This problem is undecidable in general [8]. Various decidable fragments with a PSPACE-complete reachability procedure have been studied [8, 12, 14]. The basic reachability procedure involves computing sets of reachable configurations of the UTA stored as constraints which are popularly called as *zones* [9]. A zone is a set of valuations given by a conjunction of atomic constraints  $x \triangleleft c$ ,  $c \triangleleft x$ ,  $x - y \triangleleft c$  and  $c \triangleleft x - y$  with  $c \in \mathbb{N}$  and  $x, y \in X$ . For example  $(x - y \leq 5) \wedge (2 < x)$  is a zone. Given a state-zone pair  $(q, Z)$  (henceforth called a *node*) and a transition  $t := (q, g, up, q')$ , the set of valuations  $Z_t := \{up(v) + \delta \mid v \in Z, v \models g, up(v) \geq 0, \delta \geq 0\}$  is a zone. This is the set of valuations obtained from the  $v$  in  $Z$  that satisfy the guard  $g$  of the transition, get updated to  $up(v)$  and then undergo a delay  $\delta$ . The initial node  $(q_0, Z_0)$  is obtained by delay from the initial configuration:  $Z_0 := \{v_0 + \delta \mid \delta \geq 0\}$  is a zone. This lays the foundation for a reachability procedure: start with the initial node  $(q_0, Z_0)$ ; from each node  $(q, Z)$  that is freshly seen, explore the transitions  $t := (q, g, up, q')$  out of  $q$  to compute resulting nodes  $(q', Z_t)$ . If a pair  $(q, Z)$  with  $q \in F$  is visited then the accepting state is reachable in the UTA. This naïve zone enumeration might not terminate [9]. For termination, *simulations* between zones are used.

A simulation relation on the UTA semantics is a preorder relation (in other words, a reflexive and transitive relation)  $(q, v) \sqsubseteq (q, v')$  between configurations having the same state such that the relation is preserved (1) on delay:  $(q, v + \delta) \sqsubseteq (q, v' + \delta)$  for all  $\delta \in \mathbb{R}_{\geq 0}$  and (2) on actions: if  $(q, v) \xrightarrow{t} (q_1, v_1)$ , then  $(q, v') \xrightarrow{t} (q_1, v'_1)$  with  $(q_1, v_1) \sqsubseteq (q_1, v'_1)$  for all  $t = (q, g, up, q_1)$ . This relation gets naturally lifted to zones:  $(q, Z) \sqsubseteq (q, Z')$  if for all  $v \in Z$  there exists a  $v' \in Z'$  such that  $(q, v) \sqsubseteq (q, v')$ . Intuitively, when  $(q, Z) \sqsubseteq (q, Z')$ , all sequences of transitions enabled from  $(q, Z)$  are enabled from  $(q, Z')$ . Therefore, all control states reachable from  $(q, Z)$  are reachable from  $(q, Z')$ . This allows for an optimization in the zone enumeration: a fresh node  $(q, Z)$  is not explored if there is an already visited node  $(q, Z')$  with  $(q, Z) \sqsubseteq (q, Z')$ . A simulation  $\sqsubseteq$  is said to be *finite* if in every sequence

of the form  $(q, Z_0), (q, Z_1), \dots$  there are two nodes  $(q, Z_i)$  and  $(q, Z_j)$  with  $i < j$  such that  $(q, Z_j) \sqsubseteq (q, Z_i)$ . Using a finite simulation in the reachability procedure ensures termination. Various finite simulations have been studied in the literature, the most prominent being *LU-simulation* [4, 22, 13] and more recently the  $\mathcal{G}$ -simulation [14]. In addition to ensuring termination, one needs simulations which can quickly prune the search. One main focus of research in timed automata reachability has been in finding finite simulations which are efficient in pruning the search.

In a previous work [14], we introduced a new simulation relation for UTA, called the  $\mathcal{G}$ -simulation. This relation is parameterized by a set of constraints  $\mathcal{G}(q)$  associated to every state  $q$  of the automaton. The sets  $\mathcal{G}(q)$  are identified based on the transition sequences from  $q$ . We now present the basic definitions and properties of  $\mathcal{G}$ -simulation. The presentation differs from [14], but the essence of the technical content is the same.

► **Definition 2** (*G-preorder*). *Given a finite or infinite set of constraints  $G$ , we say  $v \sqsubseteq_G v'$  if for every  $\delta \geq 0$ , and every  $\varphi \in G$ :  $v + \delta \models \varphi$  implies  $v' + \delta \models \varphi$ .*

*We simply write  $\sqsubseteq_\varphi$  instead of  $\sqsubseteq_{\{\varphi\}}$  when  $G = \{\varphi\}$  is a singleton set.*

Directly from the definition of  $\sqsubseteq_G$ , we get that the relation  $\sqsubseteq_G$  is a preorder. The definition also entails the following useful property: when  $v \sqsubseteq_G v'$ ,  $v \models \varphi$  implies  $v' \models \varphi$  for all  $\varphi \in G$ . This is a first step towards getting a simulation on the UTA semantics. It says that all guards that  $v$  satisfies are satisfied by  $v'$ , and hence all transitions enabled at  $v$  will be enabled at  $v'$  provided the transition guards are present in  $G$ . Valuations get updated on transitions and this property needs to be preserved over these updates. This motivates the following definition. It gives a constraint  $\psi$  such that  $v \sqsubseteq_\psi v'$  will imply  $up(v) \sqsubseteq_\varphi up(v')$ .

► **Definition 3**. *Given an update  $up$  and a constraint  $\varphi$ , we define  $up^{-1}(\varphi)$  to be the constraint resulting by simultaneously substituting  $up_x$  for  $x$  in  $\varphi$ :  $up^{-1}(\varphi) := \varphi[up_x/x, \forall x \in X]$ .*

For example, for  $\varphi = x - y \triangleleft c$ ,  $up^{-1}(x - y \triangleleft c) = up_x - up_y \triangleleft c$ . Similarly,  $up^{-1}(x \triangleleft c) = up_x \triangleleft c$  and  $up^{-1}(c \triangleleft x) = c \triangleleft up_x$ . Note that,  $up^{-1}(\varphi)$  need not be in the syntax defined by the grammar for constraints. But, it can be easily rewritten to an equivalent constraint satisfying this syntax. For example: consider the constraint  $x - y \triangleleft c$  and the update  $up_x = z + d$  and  $up_y = y$ , then  $up^{-1}(\varphi) = z + d - y \triangleleft c$ , which is not syntactically a constraint. However, it is equivalent to the constraint  $z - y \triangleleft c - d$ . If  $c - d < 0$ , we further rewrite as  $d - c \triangleleft y - z$ . It is also useful to note that  $up^{-1}(\varphi)$  may sometimes yield constraints equivalent to  $\top$  or  $\perp$ . For example: if  $\varphi = x \triangleleft c$  and  $up_x = d$  with  $d > c$ , then the constraint  $up^{-1}(\varphi)$  is equivalent to  $\perp$ , similarly, if  $d < c$  then  $up^{-1}(\varphi)$  is equivalent to  $\top$ .

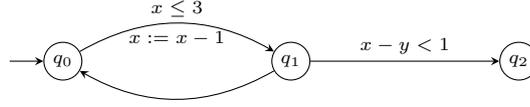
► **Lemma 4**. *Given a constraint  $\varphi$ , an update  $up$  and two valuations  $v, v'$  such that  $up(v) \geq 0$  and  $up(v') \geq 0$ , if  $v \sqsubseteq_{up^{-1}(\varphi)} v'$  then  $up(v) \sqsubseteq_\varphi up(v')$ .*

► **Definition 5** (*G-maps*). *Let  $\mathcal{A} = (Q, X, q_0, T, F)$  be a UTA. A  $\mathcal{G}$ -map  $\mathcal{G}_{\mathcal{A}}$  for UTA  $\mathcal{A}$  is a tuple  $(\mathcal{G}_{\mathcal{A}}(q))_{q \in Q}$  with each  $\mathcal{G}_{\mathcal{A}}(q)$  being a set of atomic constraints, such that the following conditions are satisfied. For every transition  $(q, g, up, q') \in T$ :*

- every atomic constraint of  $g$  belongs to  $\mathcal{G}_{\mathcal{A}}(q)$ ,
- $up^{-1}(0 \leq x) \in \mathcal{G}_{\mathcal{A}}(q)$  for every  $x \in X$ ,
- $up^{-1}(\varphi) \in \mathcal{G}_{\mathcal{A}}(q)$  for every  $\varphi \in \mathcal{G}_{\mathcal{A}}(q')$  (henceforth called the propagation criterion)

*When the UTA  $\mathcal{A}$  is clear from the context, we write  $\mathcal{G}$  instead of  $\mathcal{G}_{\mathcal{A}}$ .*

The propagation criterion allows to maintain the property described after Definition 2 even after the update occurring at transitions, and leads to a simulation relation on the configurations of the corresponding UTA, thanks to Lemma 4.



$$\begin{array}{l}
 \mathcal{G}(q_0) = \{x \leq 3, 1 \leq x\} \quad \{ \dots, x - y < 2 \} \quad \{ \dots, x \leq 4, 2 \leq x \} \quad \{ \dots, x - y < 3 \} \\
 \mathcal{G}(q_1) = \{x - y < 1\} \quad \rightarrow \{ \dots, x \leq 3, 1 \leq x \} \rightarrow \{ \dots, x - y < 2 \} \rightarrow \{ \dots, x \leq 4, 2 \leq x \} \rightarrow \dots \\
 \mathcal{G}(q_2) = \{ \} \quad \quad \quad \{ \} \quad \quad \quad \{ \} \quad \quad \quad \{ \}
 \end{array}$$

■ **Figure 1** Example automaton for which the  $\mathcal{G}$ -map computation of [14] does not terminate.

► **Definition 6** ( $\mathcal{G}$ -simulation). *Given a  $\mathcal{G}$ -map  $\mathcal{G}$ , the relation  $\sqsubseteq_{\mathcal{G}}$  on the UTA semantics defined as  $(q, v) \sqsubseteq_{\mathcal{G}} (q', v')$  whenever  $q = q'$  and  $v \sqsubseteq_{\mathcal{G}(q)} v'$ , is called the  $\mathcal{G}$ -simulation.*

In general, an automaton may not have *finite*  $\mathcal{G}$ -maps due to the propagation criterion generating more and more constraints. When a  $\mathcal{G}$ -map is finite, there is an algorithm to check  $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$ . The fewer the constraints in a  $\mathcal{G}(q)$ , the larger is the simulation  $\sqsubseteq_{\mathcal{G}(q)}$  (c.f. Definition 2). Hence there is more chance of getting  $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$  which in turn makes the enumeration more efficient. Moreover, fewer constraints in  $\mathcal{G}(q)$  give a quicker simulation test  $(q, Z) \sqsubseteq_{\mathcal{G}(q)} (q, Z')$ . The goal therefore is to get a  $\mathcal{G}$ -map as small as possible. Notice that if  $\mathcal{G}_1$  and  $\mathcal{G}_2$  are  $\mathcal{G}$ -maps, then the map  $\mathcal{G}_{min}$  defined as  $\mathcal{G}_{min}(q) := \mathcal{G}_1(q) \cap \mathcal{G}_2(q)$  is also a  $\mathcal{G}$ -map. A static analysis of the automaton to get a  $\mathcal{G}$ -map is presented in [14]. The analysis performs an iterative fixed-point computation which gives the smallest  $\mathcal{G}$ -map (for the pointwise inclusion order) whenever it terminates. A procedure to detect if the fixed-point iteration will terminate at all is also given in [14].

### 3 A new static analysis with reduced propagation of constraints

In this section we give a refined propagation criterion, which cuts short certain propagations. We start with a motivating example. Figure 1 presents an automaton and illustrates the fixed-point iteration computing the smallest  $\mathcal{G}$ -map. Identity updates (like  $y := y$ ) are not explicitly shown. Only the newly added constraints at each step are depicted. The first step adds constraints that meet the first two conditions of Definition 5. Note that  $up^{-1}(0 \leq y)$  is  $0 \leq y$  which is semantically equivalent to  $\top$ . So we do not add it explicitly to the  $\mathcal{G}$ -maps. Consider two transitions  $(q_0, v) \xrightarrow{t} (q_1, up(v))$  and  $(q_0, v') \xrightarrow{t} (q_1, up(v'))$  with  $t = (q_0, x \leq 3, x := x - 1, q_1)$ , and  $up$  being  $x := x - 1$ . Suppose we require  $up(v) \sqsubseteq_{x-y < 1} up(v')$ . By Definition 2, we need to satisfy the condition: if  $up(v) \models x - y < 1$ , then  $up(v') \models x - y < 1$ . Rewriting in terms of  $v$ : if  $v(x) - 1 - v(y) < 1$ , then  $v'(x) - 1 - v'(y) < 1$ . In other words, we need: if  $v \models x - y < 2$ , then  $v' \models x - y < 2$ . This is achieved by adding  $x - y < 2$ , the constraint  $up^{-1}(x - y < 1)$ , to  $\mathcal{G}(q_0)$  in the second step. This is the essence of the propagation criterion of Definition 5, which asks that for each  $\varphi \in \mathcal{G}(q_1)$ , we have  $up^{-1}(\varphi) \in \mathcal{G}(q_0)$ . The fixed-point computation iteratively ensures this criterion for each edge of the automaton. As illustrated, the computation does not terminate in Figure 1. There are three sources of increasing constants: (1)  $x \leq 3, x \leq 4, \dots$ , (2)  $1 \leq x, 2 \leq x, \dots$  and (3)  $x - y < 1, x - y < 2, \dots$ .

We claim that this conservative propagation is unnecessary to get the required simulation. Suppose  $v \sqsubseteq_{\mathcal{G}(q_0)} v'$  and  $(q_0, v) \xrightarrow{t} (q_1, up(v))$ , with  $t := (q_0, x \leq 3, x := x - 1, q_1)$ . Since  $t$  is enabled at  $v$ , we have  $v(x) \leq 3$ , hence  $v'(x) \leq 3$  since guard  $x \leq 3$  is present in  $\mathcal{G}(q_0)$ . We get  $v, v' \models x - y \leq 3$  as  $y \geq 0$  for all valuations. The presence of  $x - y < 4, x - y < 5, \dots$  at  $\mathcal{G}(q_0)$  is useless as both  $v, v'$  already satisfy these guards. Stopping the propagation of  $x - y < 3$  from  $\mathcal{G}(q_1)$  will cut the infinite propagation due to (3). A similar reasoning cuts

■ **Table 1** Cases where  $up^{-1}(\varphi)$  can be eliminated or replaced by a constraint with a smaller constant. We write  $\triangleleft$  and  $\triangleleft_1$  to insist that the operator  $\triangleleft$  need not be same as the operator  $\triangleleft_1$ .

	$up^{-1}(\varphi)$	$g$ contains	$pre(\varphi, g, up)$
1.	$x \triangleleft d$	$x \triangleleft_1 c$	$\top$
2.	$d \triangleleft x$	$x \triangleleft_1 c$ with $c < d$	$c \leq x$
3.	$x - y \triangleleft d$ or $d \triangleleft x - y$	$x \triangleleft_1 c$ or $x - y \triangleleft_1 c$ or $e \triangleleft_1 x - y$ s.t. $c < d < e$	$\top$

the propagation of  $x \leq 3$  from  $\mathcal{G}(q_1)$  and stops (1). The remaining source (2) is trickier, but it can still be eliminated. Here is the main idea. Consider a constraint  $3 \leq x \in \mathcal{G}(q_0)$  which propagates unchanged to  $\mathcal{G}(q_1)$  and then back to  $\mathcal{G}(q_0)$  as  $up^{-1}(3 \leq x) = 4 \leq x$ . This propagation can be cut since  $v \sqsubseteq_{3 \leq x} v'$  already ensures  $v \sqsubseteq_{4 \leq x} v'$  for the valuations that are *relevant*: the ones that satisfy the guard  $x \leq 3$  of  $t$ . Indeed,  $v, v' \models x \leq 3$  and  $v \sqsubseteq_{3 \leq x} v'$  implies  $v(x) \leq v'(x)$  which in turn implies  $v \sqsubseteq_{4 \leq x} v'$ . Overall, it can be shown that  $\mathcal{G}(q_0) = \{x \leq 3, 3 \leq x, x - y < 2, x - y < 3\}$  and  $\mathcal{G}(q_1) = \{x - y < 1\} \cup \mathcal{G}(q_0)$  suffices for the  $\mathcal{G}$ -simulation.

**Taking guards into account for propagations.** The propagation criterion of Definition 5 which is responsible for non-termination, is oblivious to the guard that is present in the transition. We will now present a new propagation criterion that takes the guard into account and cuts out certain irrelevant constraints. Consider a transition  $(q, g, up, q')$  and a constraint  $\varphi \in \mathcal{G}(q')$ . All we require is a constraint  $\psi \in \mathcal{G}(q)$  such that  $v \sqsubseteq_{\psi} v'$  and  $v \models g$  implies  $up(v) \sqsubseteq_{\varphi} up(v')$ . The additional “and  $v \models g$ ” was missing in the intuition behind the previous propagation. Of course, setting  $\psi := up^{-1}(\varphi)$  is sufficient. However, the goal is to either eliminate the need for  $\psi$  or find a  $\psi$  with a smaller constant compared to  $up^{-1}(\varphi)$ . We will see that in many cases, we can even get the former, when we plug in the “and  $v \models g$ ”.

► **Definition 7** (pre of an atomic constraint  $\varphi$  under a “guard-update” pair  $(g, up)$ ). *Let  $(g, up)$  be a pair of a guard and an update. For a constraint  $\varphi$  we define  $pre(\varphi, g, up)$  to be an atomic constraint as given by Table 1, when  $g$  and  $up^{-1}(\varphi)$  satisfy corresponding conditions. When the conditions of Table 1 do not apply,  $pre(\varphi, g, up) = up^{-1}(\varphi)$ .*

*For a set of constraints  $\mathcal{G}$ , we define  $pre(\mathcal{G}, g, up)$  to be the set  $\bigcup_{\varphi \in \mathcal{G}} \{pre(\varphi, g, up)\}$ .*

Our aim is to replace the  $up^{-1}(\varphi)$  in the older propagation criterion with  $pre(\varphi, g, up)$ . Before showing the correctness of this approach, we state a useful lemma that follows directly from the definition of  $\mathcal{G}$ -simulation.

► **Lemma 8.** *Let  $v, v'$  be valuations.*

- $v \sqsubseteq_{x \triangleleft d} v'$  iff either  $v \not\models x \triangleleft d$  or  $v'(x) \leq v(x)$
- $v \sqsubseteq_{d \triangleleft x} v'$  iff either  $v' \models d \triangleleft x$  or  $v(x) \leq v'(x)$

Readers familiar with the  $LU$ -simulation for diagonal-free automata [22] may recognize that the above lemma is almost an alternate formulation of the  $LU$ -simulation. The lemma makes a finer distinction between  $<$  and  $\leq$  in the constraints whereas  $LU$  does not.

The next proposition allows to replace the  $up^{-1}(\varphi)$  in Definition 5 by  $pre(\varphi, g, up)$  to get smaller sets of constraints at each  $q$  that still preserve the simulation. We write  $v \sqsubseteq_g v'$  for  $v \sqsubseteq_{C_g} v'$ , where  $C_g$  is the set of atomic constraints in  $g$ .

► **Proposition 9.** *Let  $(g, up)$  be a guard-update pair,  $v, v'$  be valuations such that  $v \models g$  and  $v \sqsubseteq_g v'$ , and  $\varphi$  be an atomic constraint. Then,  $v \sqsubseteq_{pre(\varphi, g, up)} v'$  implies  $v \sqsubseteq_{up^{-1}(\varphi)} v'$ .*

**Proof.** When  $\text{pre}(\varphi, g, up) = up^{-1}(\varphi)$ , there is nothing to prove. We will now prove the theorem for the combinations given in Table 1.

(Case 1). From the hypothesis  $v \sqsubseteq_g v'$ , we get  $v \sqsubseteq_{x \triangleleft_1 c} v'$ . From the other hypothesis  $v \models g$ , we get  $v \models x \triangleleft_1 c$ . Therefore, by using the formulation of  $v \sqsubseteq_{x \triangleleft_1 c} v'$  from Lemma 8, we get  $v'(x) \leq v(x)$ . This entails  $v \sqsubseteq_{x \triangleleft d} v'$  for all upper bounded guards, once again from Lemma 8.

(Case 2). We have  $\text{pre}(\varphi, g, up) = c \leq x$  and  $c < d$ . Moreover, as guard  $g$  contains  $x \triangleleft_1 c$ , we have  $v'(x) \leq v(x)$  as in Case 1. Since  $v$  satisfies the guard, we get:  $v'(x) \leq v(x) \leq c < d$ . From Lemma 8, for such valuations,  $v \sqsubseteq_{c \leq x} v'$  implies  $v'(x) = v(x)$ . Hence  $v \sqsubseteq_{d \triangleleft x} v'$ .

(Case 3). There are sub-cases depending on whether the guard contains a non-diagonal constraint or the diagonal constraints. When the guard contains  $x \triangleleft_1 c$ , we have  $v'(x) \leq v(x) \leq c$  as above. Hence  $v'(x - y) \leq c$  and  $v(x - y) \leq c$ . Since we are given that  $c < d$ , both  $v$  and  $v'$  satisfy the diagonal constraint  $x - y \triangleleft d$  and neither of them satisfies  $d \triangleleft x - y$ . Notice that time elapse preserves the satisfaction of diagonal constraints as for every valuation  $u$ ,  $(u + d)(x - y) = u(x - y)$ . From Definition 2,  $v \sqsubseteq_\psi v'$  for a diagonal constraint  $\psi$  is satisfied if  $v \not\models \psi$  or  $v' \models \psi$ . Hence,  $v \sqsubseteq_{x - y \triangleleft d} v'$  and  $v \sqsubseteq_{d \triangleleft x - y} v'$ .

For the other sub-cases of the guard containing  $x - y \triangleleft_1 c$  or  $e \triangleleft_1 x - y$ , the hypotheses  $v \models g$ ,  $v \sqsubseteq_g v'$  and the fact that  $c < d < e$  ensure the same effect, that either  $v$  does not satisfy the diagonal constraint  $up^{-1}(\varphi)$  or  $v'$  does. Hence, by definition  $v \sqsubseteq_{up^{-1}(\varphi)} v'$ . ◀

► **Definition 10** (Reduced  $\mathcal{G}$ -maps). A  $\mathcal{G}$ -map is said to be reduced if for every transition  $(q, g, up, q')$ :

- every atomic constraint of  $g$  belongs to  $\mathcal{G}(q)$ ,
- $\text{pre}(0 \leq x, g, up) \in \mathcal{G}(q)$  for every  $x \in X$ , and
- $\text{pre}(\varphi, g, up) \in \mathcal{G}(q)$  for every  $\varphi \in \mathcal{G}(q')$  (reduced propagation)

Recall the definition of  $\mathcal{G}$ -simulation of Definition 6. This is a relation  $\sqsubseteq_{\mathcal{G}}$  defined as  $(q, v) \sqsubseteq_{\mathcal{G}} (q', v')$  whenever  $q = q'$  and  $v \sqsubseteq_{\mathcal{G}(q)} v'$ . The next theorem says that this relation stays a simulation even when the  $\mathcal{G}$ -map is reduced.

► **Theorem 11.** Let  $(\mathcal{G}(q))_{q \in Q}$  be a reduced  $\mathcal{G}$ -map. The relation  $\sqsubseteq_{\mathcal{G}}$  is a simulation.

As in the case of (non-reduced)  $\mathcal{G}$ -maps, notice that if  $\mathcal{G}_1$  and  $\mathcal{G}_2$  are reduced  $\mathcal{G}$ -maps, the map  $\mathcal{G}_{\min}$  given by  $\mathcal{G}_{\min}(q) = \mathcal{G}_1(q) \cap \mathcal{G}_2(q)$  is a reduced  $\mathcal{G}$ -map. There is therefore a smallest reduced  $\mathcal{G}$ -map, given by the pointwise intersection of all reduced  $\mathcal{G}$ -maps.

► **Lemma 12.** The smallest reduced  $\mathcal{G}$ -map with respect to pointwise inclusion is the least fixed-point of the following system of equations:

$$\mathcal{G}(q) = \bigcup_{(q, g, up, q')} \{ \text{atomic constraints of } g \} \cup \{ \text{pre}(0 \leq x, g, up) \mid x \in X \} \cup \{ \text{pre}(\varphi, g, up) \mid \varphi \in \mathcal{G}(q') \}$$

The smallest reduced  $\mathcal{G}$ -map can be computed by a standard Kleene iteration. For every state  $q$  and every  $i \geq 0$ :

$$\begin{aligned} \mathcal{G}^0(q) &= \bigcup_{(q, g, up, q')} \{ \text{atomic constraints of } g \} \cup \{ \text{pre}(0 \leq x, g, up) \mid x \in X \} \\ \mathcal{G}^{i+1}(q) &= \mathcal{G}^i(q) \cup \bigcup_{(q, g, up, q')} \{ \text{pre}(\varphi, g, up) \mid \varphi \in \mathcal{G}^i(q') \} \end{aligned}$$

When  $\mathcal{G}^{k+1} = \mathcal{G}^k$ , a fixed-point has been found and  $\mathcal{G}^k$  is a reduced map satisfying Definition 10. Moreover,  $\mathcal{G}^k$  gives the least fixed-point to the system of equations of Lemma 12 and

hence  $\mathcal{G}^k$  is the smallest reduced  $\mathcal{G}$ -map. When  $\mathcal{G}^{i+1} \neq \mathcal{G}^i$  for all  $i$ , the least fixed-point is infinite and no reduced  $\mathcal{G}$ -map for the automaton can be finite. For instance, if in the UTA of Figure 1, the guard  $x \leq 3$  is removed, the smallest reduced  $\mathcal{G}$ -map will be infinite, and the fixed-point will continue forever, each iteration producing an  $x - y < c$  with increasing constants  $c$ .

It is not clear a priori how to detect whether the fixed-point computation will terminate, or will continue forever. For the non-reduced  $\mathcal{G}$ -maps, [14] gives an algorithm that runs the fixed-point computation (using  $up^{-1}$  instead of  $pre$ ) for a bounded number of steps and determines whether the computation will be non-terminating by looking for a certain witness. The reduced  $\mathcal{G}$ -map fixed-point is different due to Table 1, as certain propagations are disallowed (Cases 1 and 3), or truncated to a constant determined by the guard (Case 2). These optimizations are responsible for giving finite  $\mathcal{G}$ -maps even when the non-reduced  $\mathcal{G}$ -maps are infinite. This makes the termination analysis significantly more involved. We postpone this discussion to Section 5. In the next section, we identify some sufficient conditions that make the reduced  $\mathcal{G}$ -maps finite and describe how it leads to new applications. These observations throw more light on the mechanics of the reduced  $\mathcal{G}$ -computation and provide a preparation to the more technical Section 5.

#### 4 Applications of the reduced propagation

We exhibit three subclasses of UTA for which the reduced  $\mathcal{G}$ -maps are superior than the non-reduced  $\mathcal{G}$ -maps: either reduced  $\mathcal{G}$ -maps are finite whereas non-reduced  $\mathcal{G}$ -maps are not guaranteed to be finite, or when both are finite, the reduced  $\mathcal{G}$ -map gives a bigger simulation.

**Timed automata with bounded subtraction.** Timed automata with diagonal constraints and updates restricted to classic resets  $x := 0$  and subtractions  $x := x - c$  with  $c \geq 0$  have been used for modeling certain scheduling problems [12]. Reachability is undecidable for this restricted update model [8]. An important result in [12] is that reachability is decidable for a subclass called timed automata with *bounded subtraction*, and this decidability is used for answering the schedulability questions. Proof of decidability proceeds by constructing a region equivalence based on a maximum constant derived from the automaton. We prove that timed automata with bounded subtraction have finite reduced  $\mathcal{G}$ -maps. This gives an alternate proof of decidability and a zone-based algorithm using  $\mathcal{G}$ -simulation for this class of automata. This exercise also brings out the significance of reduced  $\mathcal{G}$ -maps: without the reduced computation, we cannot conclude finiteness.

► **Definition 13** (Timed Automata with Bounded Subtraction [12]). *A timed automaton with “subtraction” is an updatable timed automaton with updates restricted to the form  $x := 0$  and  $x := x - c$  for  $c \geq 0$ . Guards contain both diagonal and non-diagonal constraints.*

*A timed automaton with “bounded subtraction” is a timed automaton with subtraction such that there is a constant  $M_x$  for each clock  $x$  satisfying the following property for all its reachable configurations  $(q, v)$ : if there exists a transition  $(q, g, up, q')$  such that  $v \models g$  and  $up_x = x - c$  with  $c > 0$ , then  $v(x) \leq M_x$ .*

It is shown in [12] that reachability is decidable for timed automata with bounded subtraction when the bounds  $M_x$  are known. This definition of bounded subtraction puts a semantic restriction over timed automata. Indeed, reachability is decidable only when the bounds  $M_x$  are a priori known. The following is a syntactically restricted class of timed automata, that captures the bounded subtraction model when the bounds  $M_x$  are given.

► **Definition 14** (Timed Automata with Syntactically Bounded Subtraction). *This is a timed automaton with subtraction such that, for every transition  $(q, g, up, q')$  and clock  $x$ , if  $up_x = x - c$  with  $c > 0$  then the guard  $g$  contains an upper bound constraint  $x \triangleleft c'$  for some  $c' \in \mathbb{N}$ .*

► **Lemma 15.** *For every timed automaton with bounded subtraction  $\mathcal{A}'$  where the bound  $M_x$  for every clock  $x$  is known, there exists a timed automaton with syntactically bounded subtraction  $\mathcal{A}$  such that the runs of  $\mathcal{A}$  and  $\mathcal{A}'$  are the same.*

► **Theorem 16.** *The smallest reduced  $\mathcal{G}$ -maps are finite for timed automata with syntactically bounded subtraction.*

**Proof.** Let  $\overline{M}$  be the maximum constant appearing among the guards and updates of the given automaton. Define  $\overline{G}$  to be the (finite) set of all atomic constraints with constant at most  $\overline{M}$ . We will show that the finite map  $\mathcal{G}$  assigning  $\mathcal{G}(q) = \overline{G}$  for all  $q$  is a reduced  $\mathcal{G}$ -map. This then proves the theorem.

The first two conditions of Definition 10 are trivially true. It remains to show that  $\text{pre}(\overline{G}, g, up) \subseteq \overline{G}$  for every transition  $(q, g, up, q')$ . Choose a constraint  $\varphi \in \overline{G}$ . Note that  $\text{pre}(\varphi, g, up)$  is a constraint having a larger constant than  $\varphi$  only if  $up$  contains subtractions (since the other possible update is only a reset to 0 in this class). Thus, if  $up$  does not contain subtractions, from the construction of  $\overline{G}$  it follows that  $\text{pre}(\varphi, g, up) \subseteq \overline{G}$ . Now, if  $up_x = x - c$  for some clock  $x$  and  $c > 0$ , then  $g$  contains  $x \triangleleft_1 c_1$  by definition. If  $up^{-1}(\varphi)$  is some  $x \triangleleft d$ , then Case 1 of Table 1 gives  $\text{pre}(\varphi, g, up) = \top$ . If  $up^{-1}(\varphi)$  is  $d \triangleleft x$ , from Case 2 of the table, we have  $\text{pre}(\varphi, g, up) = c_1 \leq x$  or  $\text{pre}(\varphi, g, up) = d \triangleleft x$  with  $d \leq c_1$ , which are both present in  $\overline{G}$  by construction.

Finally, assume that  $up^{-1}(\varphi)$  is a diagonal constraint  $x - y \triangleleft d$  or  $d \triangleleft x - y$  and Case 3 of Table 1 does not apply. We have  $up_x = x - c_1$  with  $c_1 \geq 0$  and  $up_y = y - c_2$  with  $c_2 \geq 0$  (a reset for  $x$  or  $y$  is not possible). Moreover, if  $c_1 > 0$  (resp.  $c_2 > 0$ ) then  $g$  contains some  $x \triangleleft_1 c'_1$  (resp.  $y \triangleleft_2 c'_2$ ). If  $c_1 > 0$  then, since Case 3 does not apply, we get  $d \leq c'_1 \leq \overline{M}$  and  $up^{-1}(\varphi)$  belongs to  $\overline{G}$ . If  $c_1 = 0$  and  $c_2 > 0$  then the constraint  $\varphi$  is respectively  $x - y \triangleleft d + c_2$  or  $d + c_2 \triangleleft x - y$ . Since  $0 \leq d < d + c_2 \leq \overline{M}$ , the constraint  $up^{-1}(\varphi)$  is already in  $\overline{G}$ . ◀

Lemma 15 and Theorem 16 give an alternate proof of decidability and more importantly a zone based algorithm with optimized simulations for this model. The definition of timed automata with bounded subtraction can be seamlessly extended to include updates  $x := y - c$  where  $c \geq 0$  and  $x, y$  are potentially different clocks. Definition 14, Lemma 15 and Theorem 16 can suitably be modified to use  $y \triangleleft c'$  instead of  $x \triangleleft c'$ . This preserves the decidability, with similar proofs, even for this extended class.

**Clock bounded reachability.** Inspired by Theorem 16, we consider the problem of clock-bounded reachability: given UTA and a bound  $B \geq 0$ , does there exist an accepting run  $(q_0, v_0) \rightarrow (q_1, v_1) \rightarrow \dots (q_n, v_n)$  where  $v_i(x) \leq B$  for all  $i$  and all clocks  $x$ ? This problem is decidable for the entire class of UTA. The algorithm starts with a modified zone enumeration: each new zone is intersected with  $\bigwedge_x x \leq B$  before further exploration. This way, only the reachable configurations within the given bound are stored. The number of bounded zones is finite. Hence the enumeration will terminate without the use of any simulations. On the other hand, for efficiency, it is useful to prune the search through simulations. To use  $\mathcal{G}$ -simulation, we need a finite  $\mathcal{G}$ -map. Since we are interested in clock bounded reachability, we can inject the additional guard  $\bigwedge_x x \leq B$  in all transitions. The following theorem says that for such automata, the reduced  $\mathcal{G}$ -map will be finite. This is not true with non-reduced  $\mathcal{G}$ -maps. For instance, consider a modification of the automaton in Figure 1 with all transitions having  $x \leq 3 \wedge y \leq 3$ . This does not help cutting any of the three sources of infinite propagation that have been discussed in the text below the figure.

► **Theorem 17.** *Suppose every transition of a UTA has a guard containing an upper constraint  $x \triangleleft c$  for every clock. The reduced  $\mathcal{G}$ -map for such a UTA is finite.*

**UTA with finite non-reduced  $\mathcal{G}$ -maps.** Given a finite set of atomic constraints  $G$ , the algorithm for  $Z \sqsubseteq_G Z'$  first divides  $G$  as  $G^{nd} \cup G^d$  where  $G^{nd}$  and  $G^d$  are respectively the subsets of non-diagonal and diagonal constraints in  $G$ . From  $G^{nd}$ , two functions  $L: X \mapsto \mathbb{N} \cup \{-\infty\}$  and  $U: X \mapsto \mathbb{N} \cup \{-\infty\}$  are defined:  $L(x) = \max\{c \mid c \triangleleft x \in G^{nd}\}$  and  $U(x) = \max\{c \mid x \triangleleft c \in G^{nd}\}$ . When there is no  $c \triangleleft x$ ,  $L(x) = -\infty$ . Similarly for  $U(x)$ . Denote these functions as  $L(G)$  and  $U(G)$ . Once  $G$  is rewritten as  $L(G), U(G)$  and  $G^d$ , [14] gives a procedure to compute  $Z \sqsubseteq_G Z'$ .

For two  $\mathcal{G}$ -maps  $\mathcal{G}_1$  and  $\mathcal{G}_2$  we write  $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$  if for every  $q$  and every clock  $x$ ,  $L(\mathcal{G}_2(q))(x) \leq L(\mathcal{G}_1(q))(x)$  and  $U(\mathcal{G}_2(q))(x) \leq U(\mathcal{G}_1(q))(x)$ . We write  $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$  if  $\mathcal{G}_2(q)^d \subseteq \mathcal{G}_1(q)^d$  for every  $q$ . It can be shown that for two  $\mathcal{G}$ -maps  $\mathcal{G}_1$  and  $\mathcal{G}_2$  with  $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$  and  $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$ , the  $\mathcal{G}_2$ -simulation is bigger than the  $\mathcal{G}_1$ -simulation (using Lemma 8 for non-diagonals and the direct Definition 2 for diagonals). The following theorem asserts that when the non-reduced  $\mathcal{G}$ -map is finite, the reduced  $\mathcal{G}$ -map is finite and it induces a bigger simulation. The proof of this theorem proceeds by showing that every upper constraint  $x \triangleleft c$  and diagonal constraint added by the reduced propagation is also added by the non-reduced propagation, and for every lower constraint  $c \triangleleft x$  in the reduced  $\mathcal{G}$ , there is some  $c' \triangleleft' x$  in the non-reduced  $\mathcal{G}$  with  $c \leq c'$ .

► **Theorem 18.** *When the smallest (non-reduced)  $\mathcal{G}$ -map  $\mathcal{G}_1$  is finite, the smallest reduced  $\mathcal{G}$ -map  $\mathcal{G}_2$  is also finite. Moreover,  $LU(\mathcal{G}_2) \leq LU(\mathcal{G}_1)$  and  $\mathcal{G}_2^d \subseteq \mathcal{G}_1^d$ .*

## 5 Termination of the reduced propagation

We present an algorithm and discuss the complexity for the problem of deciding whether the smallest reduced  $\mathcal{G}$ -map of a given automaton is finite. Briefly, we present a large enough bound  $B$  such that if the fixed point iteration does not terminate in  $B$  steps, it will never terminate and hence the smallest reduced  $\mathcal{G}$ -map given by the least fixed-point is infinite.

Let us first assume that there are no strict inequalities in the atomic constraints present in guards. For the termination analysis, we can convert all strict inequalities  $<$  to weak inequalities  $\leq$ . The reduced propagation does not modify the nature of the inequality except in Case 2 of Table 1 where strict may change to weak. Any propagation in the original automaton is preserved in the converted automaton with the same constants and vice-versa. Hence the  $\mathcal{G}$ -map computation terminates in one iff it terminates in the other. We denote by  $c_\varphi$  the constant of an atomic constraint  $\varphi$ .

Let  $\mathcal{A} = (Q, X, q_0, T, F)$  be some UTA. Let  $M = \max\{c \mid c \text{ occurs in some guard of } \mathcal{A}\}$  and  $L = \max\{|d| \mid d \text{ occurs in some update of } \mathcal{A}\}$ . Let  $\mathcal{G}$  be the smallest reduced  $\mathcal{G}$ -map computed by the least fixed-point of the equations in Lemma 12. We can show that this fixed-point computation does not terminate iff a constraint with a large constant is added to some  $\mathcal{G}(q)$ .

► **Proposition 19.** *The reduced  $\mathcal{G}$ -map computation does not terminate iff for some state  $q$ , there is an atomic constraint  $\varphi \in \mathcal{G}(q)$  with a constant  $c_\varphi > N = \max(M, L) + 2L|Q||X|^2$ .*

For the analysis, we make use of strings of the form  $x \leq$ ,  $\leq x$ ,  $x - y \leq$  and  $\leq x - y$  where  $x, y \in X$  and call them *contexts*. Given a context  $\bar{\varphi}$  and a constant  $c$ , we denote by  $\bar{\varphi}[c]$  the atomic constraint obtained by plugging the constant into the context.

In the proof, we shall use the notion of propagation sequence, which is a sequence  $(q_i, \bar{\varphi}_i[c_i]) \rightarrow (q_{i+1}, \bar{\varphi}_{i+1}[c_{i+1}]) \rightarrow \dots \rightarrow (q_j, \bar{\varphi}_j[c_j])$  such that for all  $i \leq k < j$  we have  $\bar{\varphi}_{k+1}[c_{k+1}] = \text{pre}(\bar{\varphi}_k[c_k], g_k, up_k)$  for some transition  $(q_{k+1}, g_k, up_k, q_k)$  of  $\mathcal{A}$ .

**Proof of Proposition 19.** The left to right implication of Proposition 19 is clear. Conversely, assume that  $\bar{\varphi}[c] \in \mathcal{G}(q)$  for some  $(q, \bar{\varphi}[c])$  with  $c > \max(M, L) + 2L|Q||X|^2$ . Consider the smallest  $n \geq 0$  such that  $\bar{\varphi}[c] \in \mathcal{G}^n(q)$ . There is a propagation sequence  $\pi = (q_0, \bar{\varphi}_0[c_0]) \rightarrow (q_1, \bar{\varphi}_1[c_1]) \rightarrow \dots \rightarrow (q_n, \bar{\varphi}_n[c_n])$  such that  $\bar{\varphi}_0[c_0] \in \mathcal{G}^0(q_0)$  and  $(q_n, \bar{\varphi}_n[c_n]) = (q, \bar{\varphi}[c])$ . Notice that  $\bar{\varphi}_i[c_i] \in \mathcal{G}^i(q_i)$  for all  $0 \leq i \leq n$ . We first show that the propagation sequence  $\pi$  contains a positive cycle with large constants.

► **Lemma 20.** *We can find  $0 < i < j \leq n$  such that  $(q_i, \bar{\varphi}_i) = (q_j, \bar{\varphi}_j)$ ,  $c_i < c_j$  and  $\max(M, L) < c_k$  for all  $i \leq k \leq j$ .*

**Proof.** First, since  $\bar{\varphi}_0[c_0] \in \mathcal{G}^0(q_0)$  we have  $0 \leq c_0 \leq \max(M, L)$ . We consider the last occurrence of a small constant in the propagation sequence. More precisely, we define  $m = \max\{k \mid 0 \leq k < n \wedge c_k \leq \max(M, L)\}$ . Hence,  $c_k > \max(M, L)$  for all  $m < k \leq n$ .

Notice that, for  $m < k < n$ , the constraint in the sequence cannot switch from an upper diagonal to a lower diagonal and vice-versa. Indeed, assume that  $\bar{\varphi}_k[c_k] = (x - y \leq c_k)$  and  $\bar{\varphi}_{k+1}[c_{k+1}] = (c_{k+1} \leq y' - x')$ . Then the update  $up_k$  contains  $x := x' + d$ ,  $y := y' - e$  with  $c_{k+1} = d + e - c_k$ . This is a contradiction with  $d, e \leq L$  and  $c_k, c_{k+1} > L$ . Similarly, we can show that an upper (resp. lower) diagonal constraint cannot switch to a lower (resp. upper) non-diagonal constraint. On the other hand, it is possible to switch once from an upper (resp. lower) diagonal constraint to an upper (resp. lower) non-diagonal constraint.

The other remark is that  $|c_{k+1} - c_k| \leq 2L$  for all  $m \leq k < n$ . Since  $c_m \leq \max(M, L)$  and  $c_n > \max(M, L) + 2L|Q||X|^2$ , we find an increasing sequence  $m < i_1 < i_2 < \dots < i_\ell \leq n$  with  $c_{i_1} < c_{i_2} < \dots < c_{i_\ell}$  and  $\ell > |Q||X|^2$ . As noticed above, the  $\bar{\varphi}_k$  are either all upper constraints or all lower constraints, hence the set  $\{(q_k, \bar{\varphi}_k) \mid m < k \leq n\}$  contains at most  $|Q||X|^2$  elements ( $|X|$  for non-diagonals and  $|X|(|X| - 1)$  for diagonals). Therefore, we find  $i, j \in \{i_1, \dots, i_\ell\}$  such that  $i < j$  and  $(q_i, \bar{\varphi}_i) = (q_j, \bar{\varphi}_j)$ . Recall that  $c_k > \max(M, L)$  for all  $m < k \leq n$ . ◀

The next step is to show that a positive cycle with large constants can be iterated resulting in larger and larger constants.

► **Lemma 21.** *Let  $(q_i, \bar{\varphi}_i[c_i]) \rightarrow (q_{i+1}, \bar{\varphi}_{i+1}[c_{i+1}]) \rightarrow \dots \rightarrow (q_j, \bar{\varphi}_j[c_j])$  be a propagation sequence with  $(q_i, \bar{\varphi}_i) = (q_j, \bar{\varphi}_j)$ ,  $\delta = c_j - c_i > 0$  and  $M < c_k$  for all  $i \leq k \leq j$ . Then,  $(q_i, \bar{\varphi}_i[c_i + \delta]) \rightarrow (q_{i+1}, \bar{\varphi}_{i+1}[c_{i+1} + \delta]) \rightarrow \dots \rightarrow (q_j, \bar{\varphi}_j[c_j + \delta])$  is also a propagation sequence.*

This allows to conclude the proof of Proposition 19. Using Lemma 20 we obtain from  $\pi$  a positive cycle with large constants. This cycle can be iterated forever thanks to Lemma 21. We deduce that  $\bar{\varphi}_i[c_i + k\delta] \in \mathcal{G}^i(q_i)$  for all  $k \geq 0$  and the reduced  $\mathcal{G}$ -computation does not terminate. ◀

**Algorithm to detect termination.** Proposition 19 gives a termination mechanism: run the fixed-point computation  $\mathcal{G}^0, \mathcal{G}^1, \dots$ , stop if either it stabilises with  $\mathcal{G}^n = \mathcal{G}^{n+1}$  or if we add some constraint  $\varphi \in \mathcal{G}^n(q)$  with  $c_\varphi > N$ . The number of pairs  $(q, \varphi)$  with  $c_\varphi \leq N$  is  $2N|Q||X|^2$  (the factor 2 is for upper or lower constraints). Therefore, the fixed-point computation stops after at most  $2N|Q||X|^2$  steps and the total computation time is  $\text{poly}(M, L, |Q|, |X|)$ . If the constants occurring in guards and updates of the UTA  $\mathcal{A}$  are encoded in unary, the static analysis terminates in time  $\text{poly}(|\mathcal{A}|)$ . If the constants are encoded

in binary, (non-)termination of the  $\mathcal{G}$ -computation can be detected in  $\text{NPSpace} = \text{PSPACE}$ : it suffices to search for a propagation sequence  $(q_0, \varphi_0) \rightarrow (q_1, \varphi_1) \rightarrow \dots \rightarrow (q_n, \varphi_n)$  such that  $\varphi_0 \in \mathcal{G}^0(q_0)$  and  $c_{\varphi_n} > N$ . For this, we only need to store the current pair  $(q_k, \varphi_k)$ , guess some transition  $(q_{k+1}, g_k, up_k, q_k)$  of  $\mathcal{A}$ , and compute the next pair  $(q_{k+1}, \varphi_{k+1})$  with  $\varphi_{k+1} = \text{pre}(\varphi_k, g_k, up_k)$ . This can be done with polynomial space. We can also show a matching PSPACE lower-bound.

**Lower bound.** We now show that when constants are encoded in binary, deciding termination of the reduced propagation is PSPACE-hard. To do this, we give a reduction from the control-state reachability of bounded one-counter automata.

A *bounded one-counter automaton* [17, 11] is given by  $(L, \ell_0, \Delta, b)$  where  $L$  is a finite set of states,  $\ell_0$  is an initial state,  $\Delta$  is a set of transitions and  $b \geq 0$  is the global bound for the counter. Each transition is of the form  $(\ell, p, \ell')$  where  $\ell$  is the source and  $\ell'$  the target state of the transition,  $p \in [-b, +b]$  gives the update to the counter. A run of the counter automaton is a sequence  $(\ell_0, c_0) \rightarrow (\ell_1, c_1) \rightarrow \dots \rightarrow (\ell_n, c_n)$  such that  $c_0 = 0$ , each  $c_i \in [0, b]$  and there are transitions  $(\ell_i, p_i, \ell_{i+1})$  with  $c_{i+1} = c_i + p_i$ . All constants used in the automaton definition are encoded in binary. Reachability problem for this model asks if there exists a run starting from  $(\ell_0, 0)$  to a given state  $\ell_t$  with any counter value  $c_t$ . This problem is known to be PSPACE-complete [11]. We will now reduce the reachability for bounded one-counter automata to the problem of checking if the fixed-point computing the smallest reduced  $\mathcal{G}$ -map terminates (i.e, whether the smallest reduced  $\mathcal{G}$ -map is finite).

From a bounded one counter automaton  $\mathcal{B} = (L, \ell_0, \Delta, b)$  we construct a UTA  $\mathcal{A}_{\mathcal{B}}$ . States of  $\mathcal{A}_{\mathcal{B}}$  are  $L \cup \{\ell'_0, \ell'_t\}$  where  $\ell'_0$  and  $\ell'_t$  are new states not in  $L$ . There are two clocks  $x, y$ . For each transition  $(\ell, p, \ell')$  of  $\mathcal{B}$ , there is a transition  $(\ell', g, up, \ell)$  with guard  $x \leq b \wedge y \leq 0$  and updates  $x := x - p$  and  $y := y$ . We add some extra transitions using the new states  $\ell'_0$  and  $\ell'_t$ : (1)  $\ell_0 \xrightarrow{x-y \leq 0} \ell'_0$ , (2)  $\ell'_t \rightarrow \ell_t$  and (3)  $\ell'_t \xrightarrow{x:=x, y:=y+1} \ell'_t$ .

In the reduced  $\mathcal{G}$ -map computation for  $\mathcal{A}_{\mathcal{B}}$ , the constraint  $x - y \leq 0$  is added to  $\mathcal{G}^0(\ell_0)$ . The propagation sequence starting from  $(\ell_0, x - y \leq 0)$  mimicks the runs of the counter machine  $\mathcal{B}$  with the value of the diagonal constraint  $x - y \leq c$  giving the counter value. To keep this value bounded between 0 and  $b$ , we use Case 3 of Table 1. Guard  $x \leq b$  disallows propagation of constraints  $x - y \leq d$  with  $d > b$ . But, it can allow  $d$  to go smaller and smaller, and at one point the constant becomes negative and the constraint gets rewritten:  $x - y \leq b, x - y \leq b - 1, \dots, x - y \leq 0, 1 \leq y - x, 2 \leq y - x$ , etc. The presence of the constraint  $y \leq 0$  in the guard eliminates  $1 \leq y - x, 2 \leq y - x$ , etc. once again due to Case 3.

► **Lemma 22.** *For every run  $(\ell_0, 0) \rightarrow (\ell_1, c_1) \rightarrow \dots \rightarrow (\ell_n, c_n)$  in  $\mathcal{B}$ , there is a propagation sequence  $(\ell_0, x - y \leq 0) \rightarrow (\ell_1, x - y \leq c_1) \rightarrow \dots \rightarrow (\ell_n, x - y \leq c_n)$  in  $\mathcal{A}_{\mathcal{B}}$ .*

► **Lemma 23.** *For every propagation sequence  $(\ell_0, x - y \leq 0) \rightarrow (\ell_1, x - y \leq c_1) \rightarrow \dots \rightarrow (\ell_n, x - y \leq c_n)$  in  $\mathcal{A}_{\mathcal{B}}$  with  $\ell_i \in L$  for  $0 \leq i \leq n$ , there is a run  $(\ell_0, 0) \rightarrow (\ell_1, c_1) \rightarrow \dots \rightarrow (\ell_n, c_n)$  in  $\mathcal{B}$ .*

It now remains to notice that the only transition that can generate infinitely many constraints during the propagation is the loop  $\ell'_t \rightarrow \ell'_t$ , since the other transitions between states coming from the counter automaton have a guard to cut out infinite propagations. For this to happen some constraint needs to reach  $\ell_t$ , and propagate to  $\ell'_t$  via  $\ell'_t \rightarrow \ell_t$ .

► **Proposition 24.** *The final state is reachable in the counter automaton  $\mathcal{B}$  iff the smallest reduced  $\mathcal{G}$ -map of  $\mathcal{A}_{\mathcal{B}}$  is infinite.*

■ **Table 2** # nodes is the number of nodes enumerated during a breadth-first-search; “-” denote that there was no answer for 20 minutes; N/A denotes not-applicable. Experiments were conducted on a MacBook Pro with 8 GB RAM, 2.3Ghz processor running macOS 10.14.

Model	Schedulable?	New static analysis		Static analysis of [14]	
		# nodes	time	# nodes	time
SporadicPeriodic-5	Yes	677	1.710s	-	-
SporadicPeriodic-20	No	852	1.742s	-	-
Mine-Pump	Yes	31352	7m 23.509s	-	-
<i>Flower</i> task triggering automaton: (computation time, deadline)					
(1,2), (1,2), (1,2)	No	212	0.057s	-	-
(1,10), (1,10), (1,10), (1,4)	Yes	105242	8m 57.256s	-	-
<i>Worst-case</i> task triggering automaton: (computation time, deadline)					
(1,2), (1,2), (1,2)	No	20	0.050s	-	-
(1,10), (1,10), (1,10), (1,4)	Yes	429	0.454s	-	-
12 copies of (1,20)	Yes	786	12m 5.250s	-	-
$\mathcal{A}_{gain} \times 3$	N/A	24389	7.611s	24389	12.402s
$\mathcal{A}_{gain} \times 4$	N/A	707281	14m 12.369s	707281	27m 13.540s

► **Theorem 25.** *Deciding termination of the reduced  $\mathcal{G}$ -map computation for a given UTA  $\mathcal{A}$  is in PTIME if the constants in  $\mathcal{A}$  are encoded in unary, and PSPACE-complete if the constants are encoded in binary.*

## 6 Experiments

We report on experiments conducted using the open source tool TChecker [20]. The models are given as networks of timed automata which communicate via synchronized actions. We have implemented the new static analysis discussed in Section 3. The older static analysis and zone enumeration with the  $\mathcal{G}$ -simulation were already implemented [14].

**Compositionality of static analysis.** Both these static analyses are performed individually on each component. For each local state  $q_i$  a map  $\mathcal{G}(q_i)$  is computed. During the zone enumeration the product of the automata is computed on-the-fly. Each node is of the form  $(q, Z)$  where  $q = \langle q_1, q_2, \dots, q_k \rangle$  is a tuple of local states, one from each component of the network and  $Z$  is a zone over all clocks of the network. The  $\mathcal{G}$ -map is then taken as  $\mathcal{G}(q) = \bigcup_i \mathcal{G}(q_i)$ . This approach creates a problem when there are shared clocks. A component  $i$  might update  $x$  and another component  $j \neq i$  might contain a guard with  $x$ . The  $\mathcal{G}$ -maps computed component-wise will then not give a sound simulation. In our experiments, we construct models without shared clocks.

**Benchmarks.** Our primary benchmarks are models of task scheduling problems using the Earliest-Deadline-First (EDF) policy. Each task has a computation time and a deadline. Tasks are released either periodically or via a specification given as a timed automaton. The goal is to verify if for a given set of released tasks, all of them can be finished within their deadline. Preemption of tasks is allowed. This problem has been encoded as a reachability in a network of timed automata that uses bounded subtraction [12]. The main challenge is to model preemption. Each task  $t_i$  has an associated clock  $c_i$  which is reset as soon as the task starts to execute. While  $t_i$  is running, and some other task  $t_j$  preempts  $t_i$ , the clock  $c_i$  continues to elapse time. When  $t_j$  is done, an update  $c_i := c_i - C_j$  is performed,

where  $C_j$  denotes the computation time of  $t_j$ . This way, when  $t_i$  is scheduled again, clock  $c_i$  maintains the computation time that has elapsed since it was started. Whenever the EDF scheduler has to choose between task  $t_i$  and  $t_j$ , it chooses the one which is closest to its deadline. To get this, when  $t_i$  is released, a clock  $d_i$  is reset. Task  $t_i$  is prioritized over  $t_j$  if  $D_i - d_i < D_j - d_j$  where  $D_i, D_j$  are the deadlines. We have constructed a model for the EDF scheduler based on these ideas (more details in [15]). For the experiments in Table 2, we consider some task release strategies given in the literature (SporadicPeriodic from TIMES tool, and a variant of Mine-Pump from [16]) and also create some of our own (Flower and Worst-case task triggers). The last model  $\mathcal{A}_{gain}$  is an automaton with reset-to-zero only updates illustrating the gain when both static analyses terminate.

**Comparison.** For all the EDF examples, the old static analysis did not terminate, as seen in the last two columns of Table 2. This is expected since the model contains an update of the form  $x := x - C$  which repeatedly adds guards  $x \leq K, x \leq K + C, x \leq K + 2C, \dots$ . The new static analysis cuts this out, since the update  $x := x - C$  occurs along with a guard  $x \leq D$ , making it a timed automaton with bounded subtraction. The  $\mathcal{A}_{gain}$  example runs with both the static analyses. However, the new static analysis minimizes the propagation of diagonal constraints. The time taken by the simulation test used in the zone enumeration phase is highly sensitive to the number of diagonal constraints. Fewer diagonals therefore result in a faster zone enumeration. We have also tried our new static analysis for standard benchmarks of diagonal-free timed automata and observed no gain. In these models, the distance between a clock reset and a corresponding guard (in a component automaton) is small, usually within one or two transitions. Hence resets already cut out most of the guards and the optimizations of Table 1 do not seem to help here. We expect to gain primarily in the presence of updates or diagonal constraints. We also remark that the last experiment cannot be performed on the TIMES tool which is built for scheduling problems and the previous ones cannot be modeled in other timed automata tools UPPAAL, PAT and Theta since they cannot handle timed automata with subtraction updates. Our prototype therefore subsumes existing tools in terms of modeling capability.

## 7 Conclusion

We have presented a static analysis procedure for UTA. This method terminates for a wider class of UTA compared to [14], and hence makes powerful simulations applicable to these systems. We have experimented with a prototype implementation. At a technical level, we get a unifying framework to show decidability of the reachability problem for automata with diagonal constraints and updates  $x := c$  and  $x := y + d$  that covers the decidable subclasses of [8], [12] and [14], the only known decidable classes upto our knowledge. Our framework provides a high-level technique to extend to broader classes: to show decidability, check if there is a finite reduced  $\mathcal{G}$ -map (c.f. proof of Theorem 16 and the subsequent remark). Earlier route via regions [8, 12] requires a more involved low-level reasoning to show the correctness of the region equivalence. From a practical perspective, we have a prototype with a richer modeling language and a more efficient way to handle updates than the existing real-time model checkers. As future work, we plan to engineer the prototype to make it applicable for bigger models and release the implementation and benchmarks in the public domain.

---

**References**

---

- 1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- 2 Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: A tool for schedulability analysis and code generation of real-time systems. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.
- 3 Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2003.
- 4 Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.
- 5 Mikolaj Bojanczyk and Slawomir Lasota. A machine-independent characterization of timed languages. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 7392 of *Lecture Notes in Computer Science*, pages 92–103. Springer, 2012.
- 6 Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.
- 7 Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.
- 8 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.
- 9 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.
- 10 David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
- 11 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is pspace-complete. *Inf. Comput.*, 243:26–36, 2015.
- 12 Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.
- 13 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In *International Conference on Concurrency Theory (CONCUR)*, volume 118 of *LIPICs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 14 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In *Computer Aided Verification (CAV)*, pages 41–59, Cham, 2019. Springer International Publishing.
- 15 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability for updatable timed automata made faster and more effective, 2020. [arXiv:2009.13260](https://arxiv.org/abs/2009.13260).
- 16 Thorsten Gerdsmeyer and Rachel Cardell-Oliver. Analysis of scheduling behaviour using generic timed automata. *Electronic Notes in Theoretical Computer Science*, 42:143–157, 2001. Computing: The Australasian Theory Symposium (CATS).
- 17 Christoph Haase, Joël Ouaknine, and James Worrell. Relating reachability problems in timed and counter automata. *Fundam. Inform.*, 143(3-4):317–338, 2016.
- 18 Leo Hatvani, Alexandre David, Cristina Cerschi Seceleanu, and Paul Pettersson. Adaptive task automata with earliest-deadline-first scheduling. *ECEASST*, 70, 2014.
- 19 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What’s decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.
- 20 Frédéric Herbretau and Gerald Point. TChecker. URL: <https://github.com/ticktac-project/tchecker>.

- 21 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 990–1005. Springer, 2013.
- 22 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016.
- 23 Pavel Krcál, Martin Stigge, and Wang Yi. Multi-processor schedulability analysis of preemptive real-time tasks with variable execution times. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 4763 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2007.
- 24 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
- 25 Yuki Osada, Tim French, Mark Reynolds, and Harry Smallbone. Hourglass automata. In *Games, Automata, Logic and Formal verification (GandALF)*, volume 161 of *EPTCS*, pages 175–188, 2014.
- 26 Victor Roussanaly, Ocan Sankur, and Nicolas Markey. Abstraction refinement algorithms for timed automata. In *Computer Aided Verification (CAV)*, volume 11561 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2019.
- 27 Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In *Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.
- 28 Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: a framework for abstraction refinement-based model checking. In Daryl Stewart and Georg Weissenbacher, editors, *Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 176–179, 2017. doi:10.23919/FMCAD.2017.8102257.
- 29 Sergio Yovine. Kronos: A verification tool for real-time systems. (Kronos user’s manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.



# Active Prediction for Discrete Event Systems

Stefan Haar 

INRIA, LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France  
stefan.haar@inria.fr

Serge Haddad 

LSV, ENS Paris-Saclay, CNRS, INRIA, Université Paris-Saclay, France  
haddad@lsv.fr

Stefan Schwoon 

LSV, ENS Paris-Saclay, CNRS, INRIA, Université Paris-Saclay, France  
schwoon@lsv.fr

Lina Ye 

LRI, Université Paris-Saclay, CentraleSupélec, France  
lina.ye@lri.fr

---

## Abstract

A central task in partially observed controllable system is to detect or prevent the occurrence of certain events called *faults*. Systems for which one can design a controller avoiding the faults are called *actively safe*. Otherwise, one may require that a fault is eventually detected, which is the task of *diagnosis*. Systems for which one can design a controller detecting the faults are called *actively diagnosable*. An intermediate requirement is *prediction*, which consists in determining that a fault will occur whatever the future behaviour of the system. When a system is not predictable, one may be interested in designing a controller to make it so. Here we study the latter problem, called *active prediction*, and its associated property, *active predictability*. In other words, we investigate how to determine whether or not a system enjoys the active predictability property, i.e., there exists an active predictor for the system.

Our contributions are threefold. From a semantical point of view, we refine the notion of predictability by adding two quantitative requirements: the minimal and maximal delay before the occurrence of the fault, and we characterize the requirements fulfilled by a controller that performs predictions. Then we show that active predictability is EXPTIME-complete where the upper bound is obtained via a game-based approach. Finally we establish that active predictability is equivalent to active safety when the maximal delay is beyond a threshold depending on the size of the system, and we show that this threshold is accurate by exhibiting a family of systems fulfilling active predictability but not active safety.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory; Mathematics of computing → Discrete mathematics

**Keywords and phrases** Automata Theory, Partially observed systems, Diagnosability, Predictability

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.48

**Related Version** A full version of the paper is available at [14], <https://hal.archives-ouvertes.fr/hal-02951944>.

**Funding** *Lina Ye*: This research was done while Lina Ye was on leave at MEXICO team of INRIA.

## 1 Introduction

**Monitoring faulty systems.** In monitoring faulty systems, two central tasks consist in detecting a fault that has occurred, resp. will occur, i.e. the tasks of *diagnosis* and *prediction*, respectively, based on observations. However, such tasks may be defeasible due to ambiguity (i.e. observations associated with both correct and faulty runs). In this case, one may



© Stefan Haar, Serge Haddad, Stefan Schwoon, and Lina Ye;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 48; pp. 48:1–48:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

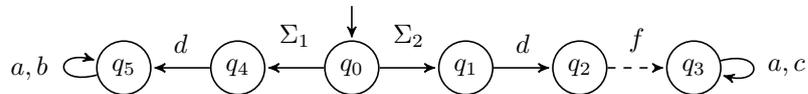
introduce a *controller* to restrict the behaviour in order to enforce diagnosis (resp. prediction) to be processed. Such a controller is called an *active diagnoser* (resp. *active predictor*). Here we focus on the existence of an active predictor, a problem called *active predictability*.

**Diagnosis.** In partially observed discrete-event systems, diagnosis was defined and studied in the seminal paper by Sampath et al [17] (see also [6, 7]). That work builds a deterministic version of the original model, a so-called *diagnoser*, that tries to detect the occurrence of faults. A system is called *diagnosable* if the diagnoser can detect every fault occurrence, possibly after some delay. As an illustration, consider the system in Figure 1, which we shall use as a running example, sometimes with different values for  $\Sigma_1$  and  $\Sigma_2$ , where  $\Sigma_1$  and  $\Sigma_2$  are subsets of events in the system. Precisely,  $\Sigma_1, \Sigma_2 \subseteq \{a, b, c, d\}$ , all of which are observable, while  $f$  represents a fault that is not directly observable. If, e.g.,  $a$  is contained in both  $\Sigma_1$  and  $\Sigma_2$ , then the system is not diagnosable because any observation  $ada^n$  may belong to a faulty run or a correct one.

The diagnosability problem is in PTIME [22], via an approach called *twin-plant construction*. When the system is not diagnosable, it may have to be redesigned, e.g. by adding further sensors to enhance observability, or by forbidding some actions. Sampath et al [16] followed the last approach, called *active diagnosis*: one strives to synthesise a controller, based on partial observations, that forces the behaviour of the system to stay within a diagnosable subset of its behaviours. For instance, if the system in Figure 1 has  $\Sigma_1 = \Sigma_2 = \{b\}$  and the controller has the right to block  $a$ , then the system is actively diagnosable.

The algorithm for the active-diagnosability problem in [16] operates in doubly exponential time. In [13], we revisited the problem using automata and game theory and established that in fact the active-diagnosability problem is EXPTIME-complete. Later on, we generalised the framework, e.g. allowing the controller to be aware of deadlocks [4]. We also studied active diagnosis for probabilistic systems [1].

In loosely related works. Chantry and Pencolé [9] proposed a planning-based approach that allows the verdict of the diagnoser to be ambiguous; the works in [8, 10, 20] studied the problem of dynamic sensor activation to ensure some observation properties. In work more closely related to ours [19], Yin and Lafortune proposed a uniform approach for synthesizing property-enforcing supervisor by mapping the considered property to a suitably-defined information state, which is applicable to a class of properties that can be expressed in terms of such information states, including safety, diagnosability, opacity and so on. Note that predictability cannot be formulated as an information state in that framework since it depends also on future behaviours of the system; its enforcement thus requires new methods.



■ **Figure 1** Running example, with unobservable events indicated by dashed lines.

**Prediction.** Several works have studied the (passive) predictability problem, i.e. where no control is involved. For instance, if  $\Sigma_1 = \{b\}$  and  $\Sigma_2 = \{c\}$  in Figure 1, then upon first seeing  $c$ , an observer can predict that a fault will necessarily occur. In [11], Genc and Lafortune introduced a diagnoser construction to derive a necessary and sufficient condition for predictability in systems modeled by regular languages. Ye, Dague, and Nouioua [18] proposed a polynomial time algorithm for predictability analysis in a centralized way and

then extend it to a distributed framework. Brandan Briones and Madalinski [5] introduced and studied two variants of predictability by defining an additional requirement about either a lower bound or an upper bound on the number of events between the fault prediction and the fault occurrence. Then Yin and Li [21] investigated the bounded predictability in the decentralized framework, and proposed a polynomial-time algorithm for its verification. Madalinski and Khomenko [15] reduce the predictability problem for a Petri net to LTL-X model checking. All these previous works focus on passive predictability.

**Our contributions.** First we refine the paradigm of prediction by allowing an observer to quantify its observations. Unlike [5] but similar to [21], our predictors can at the same time provide both lower and upper bounds on the number of observations before a fault may (resp. must) occur. For instance, upon seeing  $c$  in the previous example, an observer can not only predict that a fault will eventually happen but that it will necessarily happen between the first and the second observable event after  $c$ . In practice, if a fault prediction is issued, the reaction procedure of the system should be triggered. As such interventions may require a certain amount of time to take effect, having both lower and upper bounds are relevant performance criteria for capture such timing issues.

We then turn to the case of active prediction, where a controller tries to restrict the system's behaviour so that faults can be reliably predicted. For instance, if  $\Sigma_1 = \{a, b\}$  and  $\Sigma_2 = \{a, c\}$  in Figure 1, then faults are unpredictable, but if a controller has the right to block  $a$ , it becomes actively predictable (with the aforementioned bounds). We formalize the idea of active predictability and then propose a class of controller, called active predictor. We then show that active predictability is equivalent to the existence of an active predictor.

Next, we focus on the decision and synthesis problems, i.e. to decide whether the system is actively predictable, and if so, how to build an active predictor. In active *diagnosability* [13], the solution exploited the fact that whether a sequence of observations is ambiguous (i.e. corresponds to both faulty and correct runs) is *independent of the control* that was applied in the past. In prediction, by contrast, the eventuality of a fault occurrence in the future *depends on the control* that is going to be applied. Thus solving the active-predictability problems requires new techniques.

We establish that the decision problem is EXPTIME-complete by reducing it to a turn-based game with a Büchi objective of exponential size. A memoryless winning strategy of this game provides the main ingredient to build an active predictor. Furthermore we show that instead of solving this Büchi game (which takes quadratic time), one can equivalently in linear time (1) solve a reachability game, (2) build a safety game that depends on the winning states of the reachability game, and (3) solve it and combine the winning strategies to get a winning strategy for the Büchi game when it exists (see [14] for all details).

Finally we study the relation between the lower prediction bound  $k$  and the number of states  $n$  of the system. We establish that if  $k \geq 2^n$  then a system is  $k$ -actively predictable if and only if it is actively safe. This bound is tight since we exhibit a family of systems of size  $\mathcal{O}(n)$  such that the system is  $2^n$ -actively predictable but not actively safe.

**Organization.** In Section 2, we introduce prediction in both the uncontrollable and controllable framework and establish a class of controller called *active predictor*. The existence of such a controller is equivalent to active predictability. The construction of an active predictor (if it exists) is carried out in Section 3, providing simultaneously the solutions to the decision and synthesis problems. Section 4 complements these results by a tight analysis of complexity bounds. We conclude and give some perspectives to this work in Section 5. The missing proofs are developed in [14].

## 2 The Active Prediction Problem

As usual, for an alphabet  $\Sigma$ , we use  $\Sigma^*$  and  $\Sigma^\omega$ , to denote the finite and infinite words over  $\Sigma$ , and  $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$ . The length of a word  $\sigma \in \Sigma^*$  is denoted  $|\sigma|$ , and  $\preceq$  represents the prefix notation.

### Labeled transition systems

When dealing with discrete event systems (DES), systems are often modeled using labeled transition systems (LTS).

► **Definition 1.** A labeled transition system is a tuple  $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$  where:

- $Q$  is a set of states with  $q_0 \in Q$  the initial state;
- $\Sigma$  is a finite set of events;
- $T \subseteq Q \times \Sigma \times Q$  is a set of transitions.

We note  $q \xrightarrow{a}_{\mathcal{A}} q'$  for  $(q, a, q') \in T$ ; this transition is said to be *enabled* in  $q$ . A *run* over the infinite word  $\sigma = a_1 a_2 \dots \in \Sigma^\omega$  is a sequence of states  $(q_i)_{i \geq 0}$  with  $q_i \xrightarrow{a_{i+1}}_{\mathcal{A}} q_{i+1}$  for all  $i \geq 0$ , and we write  $q_0 \xrightarrow{\sigma}_{\mathcal{A}}$  if such a run exists. A finite run over  $\sigma \in \Sigma^*$  is defined analogously, and we write  $q \xrightarrow{\sigma}_{\mathcal{A}} q'$  if it ends at state  $q'$ . A state  $q$  is *reachable* if there exists a run  $q_0 \xrightarrow{\sigma}_{\mathcal{A}} q$  for some  $\sigma$ . The index  $\mathcal{A}$  in those relations will be omitted if unambiguous.

In order to formalize problems related to prediction, we partition  $\Sigma$  into two disjoint sets  $\Sigma_o$  and  $\Sigma_{uo}$ , the sets of *observable* and of *unobservable events*, respectively. Moreover, we distinguish a special *fault* event  $f \in \Sigma_{uo}$ . We say  $\sigma$  is *correct* if  $\sigma \in (\Sigma \setminus \{f\})^*$  (we will denote  $\Sigma \setminus \{f\}$  with the short form  $\Sigma \setminus^f$  in the following), and that  $\sigma$  is *faulty* otherwise. For  $\Sigma' \subseteq \Sigma$ , define its projection  $\mathcal{P}_{\Sigma'}(\sigma)$  inductively by:  $\mathcal{P}_{\Sigma'}(\varepsilon) = \varepsilon$ ;  $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)a$  when  $a \in \Sigma'$ , and  $\mathcal{P}_{\Sigma'}(\sigma a) = \mathcal{P}_{\Sigma'}(\sigma)$  otherwise. For the sake of simplicity, write  $\mathcal{P}$  for  $\mathcal{P}_{\Sigma_o}$ ,  $|\sigma|_o$  for  $|\mathcal{P}(\sigma)|$ ,  $|\sigma|_{\Sigma'}$  for  $|\mathcal{P}_{\Sigma'}(\sigma)|$ , and for  $a \in \Sigma$ , write  $|\sigma|_a$  for  $|\sigma|_{\{a\}}$ . When  $\sigma$  is an infinite word, its projection is the limit of the projections of its finite prefixes. This projection can be either finite or infinite. As usual the projection is extended to languages.

► **Definition 2** (Languages of an LTS). Let  $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$  be an LTS. The finite and the infinite (correct) languages of  $\mathcal{A}$  are defined by:

- $\mathcal{L}^*(\mathcal{A}) = \{ \sigma \in \Sigma^* \mid \exists q, q_0 \xrightarrow{\sigma}_{\mathcal{A}} q \}$  and  $\mathcal{L}^\omega(\mathcal{A}) = \{ \sigma \in \Sigma^\omega \mid q_0 \xrightarrow{\sigma}_{\mathcal{A}} \}$ ;
  - $\mathcal{L}_c^*(\mathcal{A}) = \{ \sigma \in (\Sigma \setminus^f)^* \mid \exists q, q_0 \xrightarrow{\sigma}_{\mathcal{A}} q \}$  and  $\mathcal{L}_c^\omega(\mathcal{A}) = \{ \sigma \in (\Sigma \setminus^f)^\omega \mid q_0 \xrightarrow{\sigma}_{\mathcal{A}} \}$
- $\mathcal{A}$  is safe if  $\mathcal{L}^*(\mathcal{A}) = \mathcal{L}_c^*(\mathcal{A})$  (i.e. no fault ever occurs).

The union of finite and infinite languages of  $\mathcal{A}$  is denoted  $\mathcal{L}^\infty(\mathcal{A}) = \mathcal{L}^*(\mathcal{A}) \cup \mathcal{L}^\omega(\mathcal{A})$ . The inverse observable projection with respect to  $\mathcal{A}$  and  $w \in \Sigma_o^*$  is defined as  $\mathcal{P}_{\mathcal{A}}^{-1}(w) = \{ \sigma \in \mathcal{L}^*(\mathcal{A}) \mid \mathcal{P}(\sigma) = w \}$ , which can be simply denoted by  $\mathcal{P}^{-1}(w)$  if there is no ambiguity. An LTS  $\mathcal{A}$  is *deterministic* if for every pair  $q \in Q, a \in \Sigma$  there is at most one  $q'$  such that  $q \xrightarrow{a}_{\mathcal{A}} q'$ . For a deterministic LTS we write  $T(q, a) = q'$  if  $q \xrightarrow{a}_{\mathcal{A}} q'$ . As is the case for classical diagnosis problems, we make two **assumptions** on  $\mathcal{A}$ :

- Liveness:  $\forall q \in Q, \exists a, q', q \xrightarrow{a}_{\mathcal{A}} q'$ .
- Convergence:  $\mathcal{L}^\omega(\mathcal{A}) \cap \Sigma^* \Sigma_{uo}^\omega = \emptyset$ .

Liveness implies that from any reachable state of an LTS, there exists at least one transition enabled in that state. Convergence guarantees that there is no infinite sequence of unobservable events. When  $\mathcal{A}$  is convergent, then for all  $\sigma \in \mathcal{L}^\omega(\mathcal{A})$ , one has  $\mathcal{P}(\sigma) \in \Sigma_o^\omega$ .

► **Example 3.** Figure 1 shows a live and convergent LTS with  $\Sigma_o = \{a, b, c, d\}$ ,  $\Sigma_{uo} = \{f\}$ ,  $\Sigma_1 \subseteq \Sigma_o$ ,  $\Sigma_2 \subseteq \Sigma_o$  and  $\Sigma_1 \cup \Sigma_2 \neq \emptyset$ . Transitions labelled by unobservable events are dashed. We also factorize transitions with same source and target states. Depending on  $\Sigma_1$  and  $\Sigma_2$ , this LTS may have different levels of predictability (see Example 7 for further explanation).

## Predictability

Intuitively, a system is predictable with respect to a fault  $f$  if in every faulty run, an observer can be certain that  $f$  is going to occur before it actually happens. Before formally defining predictability, we first present some useful notations. Given  $\sigma \in \mathcal{L}^\infty(\mathcal{A})$  and  $n \leq |\sigma|_o$ ,  $pre_n(\sigma)$  denotes the minimal (w.r.t.  $\preceq$ ) prefix of  $\sigma$  such that  $|pre_n(\sigma)|_o = n$ . As an abbreviation,  $pre(\sigma) := pre_{|\sigma|_o}(\sigma)$  removes unobservable events from the end of  $\sigma$ . For example, in the LTS of Figure 1, we have (as  $f$  is unobservable)  $pre_0(bdf) = \varepsilon$ ,  $pre_1(bdf) = b$  and  $pre(bdf) = pre_2(bdf) = bd$ . We naturally extend  $pre$  to sets of words.

An observed sequence  $w$  forbids prediction of a fault when a correct, infinite future behavior is still possible. We introduce different kinds of observed sequences.

- **Definition 4.** (*observation properties*) Let  $\mathcal{A}$  be an LTS,  $w \in \Sigma_o^*$ , and  $m \in \mathbb{N}$ . Then  $w$  is:
- surely correct in  $\mathcal{A}$  if  $pre(\mathcal{P}_{\mathcal{A}}^{-1}(w)) \cap \Sigma^* f \Sigma^* = \emptyset$ ;
  - surely faulty in  $\mathcal{A}$  if  $\mathcal{P}_{\mathcal{A}}^{-1}(w) \cap \mathcal{L}_c^*(\mathcal{A}) = \emptyset$ ;
  - ambiguous in  $\mathcal{A}$  if it is neither surely correct nor surely faulty in  $\mathcal{A}$ ;
  - $m$ -correct in  $\mathcal{A}$  if  $ww'$  is surely correct in  $\mathcal{A}$  for all  $w' \in \Sigma_o^m$ ;
  - $m$ -faulty in  $\mathcal{A}$  if  $ww'$  is surely faulty in  $\mathcal{A}$  for all  $w' \in \Sigma_o^m$ ;
  - $\omega$ -faulty in  $\mathcal{A}$  if there exists  $m \in \mathbb{N}$  such that  $w$  is  $m$ -faulty.

We now define the notion of  $k$ - $l$ -predictability, which means that the occurrence of a fault can be predicted with certainty, based on what has been observed so far, at least  $k$  observations before it does occur, and such that the fault definitely occurs before the  $l$ -th additional observation. In the sequel,  $\mathbb{N}^+$  denotes  $\mathbb{N} \setminus \{0\}$  and  $\mathbb{N}_\omega$  (resp.  $\mathbb{N}_\omega^+$ ) denotes  $\mathbb{N}$  (resp.  $\mathbb{N}^+$ ) enlarged with  $\omega$  which is an absorbing item for addition.

- **Definition 5.** (*Predictability*) Let  $\mathcal{A}$  be an LTS,  $w \in \Sigma_o^*$ ,  $k \in \mathbb{N}$ , and  $l \in \mathbb{N}_\omega^+$ .
- $w$  is  $k$ - $l$ -faulty in  $\mathcal{A}$  if  $w$  is  $k$ -correct and  $(k+l)$ -faulty in  $\mathcal{A}$ .
  - $\mathcal{A}$  is  $k$ - $l$ -predictable if for all  $\sigma f \in \mathcal{L}^*(\mathcal{A})$ ,  $\mathcal{P}(\sigma)$  has a  $k$ - $l$ -faulty prefix.

► **Remark 6.** If  $w$  is  $k$ - $l$ -faulty in  $\mathcal{A}$ , then  $w$  is also  $k'$ - $l'$ -faulty in  $\mathcal{A}$  for all  $k' \leq k$  and  $k' + l' \geq k + l$ .

As an abbreviation, we will call  $\mathcal{A}$   $k$ -predictable if it is  $k$ - $\omega$ -predictable, and simply *predictable* if it is 0-predictable. Thus, Remark 6 implies that predictability is weaker than any other notion of  $k$ - $l$ -predictability.

- **Example 7.** Consider the LTS of Figure 1:
- it is not predictable if  $\Sigma_1 \cap \Sigma_2 \neq \emptyset$ ;
  - it is 1-1-predictable and not 2-predictable if  $\Sigma_1 \cap \Sigma_2 = \emptyset$ , and both of them are not empty;
  - it is 2-1-predictable if  $\Sigma_1 = \emptyset$  and  $\Sigma_2 \neq \emptyset$ .

Proposition 8 establishes bounds for predictability in finite LTS:

- **Proposition 8.** Let  $\mathcal{A}$  be a  $k$ -predictable LTS with  $n$  states, where  $n$  is finite.
- (i)  $\mathcal{A}$  is  $k$ - $n$ -predictable.
  - (ii) If  $\mathcal{A}$  is not safe, then  $k < n$ .

### Active predictability

We suppose that  $\Sigma_o$  is partitioned into subsets  $\Sigma_c \subseteq \Sigma_o$  of *controllable* and  $\Sigma_{uco} = \Sigma_o \setminus \Sigma_c$  of *uncontrollable* actions. Intuitively, a controller may forbid a subset of the controllable actions based on the observations made so far, thereby restricting the behaviour of  $\mathcal{A}$ .

► **Definition 9** (Controlled LTS). *Let  $\mathcal{A}$  be an LTS. A controller for  $\mathcal{A}$  is a mapping  $cont : \mathcal{P}(\mathcal{L}^*(\mathcal{A})) \rightarrow 2^{\Sigma}$  such that for all  $w$ ,  $\Sigma_{uco} \cup \Sigma_{uo} \subseteq cont(w)$ . The controlled LTS  $\mathcal{A}_{cont} = \langle Q_{cont}, q_{0cont}, \Sigma, T_{cont} \rangle$  is defined as the smallest LTS satisfying:*

- $q_{0cont} = \langle \varepsilon, q_0 \rangle \in Q_{cont}$ ;
- if  $\langle w, q \rangle \in Q_{cont}$ ,  $a \in cont(w)$ , and  $q \xrightarrow{a}_{\mathcal{A}} q'$ , then  $\langle w\mathcal{P}(a), q' \rangle \in Q_{cont}$  and  $\langle w, q \rangle \xrightarrow{a}_{\mathcal{A}_{cont}} \langle w\mathcal{P}(a), q' \rangle$ .

The goal of our controllers is to make the system predictable by preserving liveness and to perform prediction at the same time. Before formally defining prediction verdicts in Definition 11, we discuss their intuitive meanings:  $\top$  means that the controller is currently unable to predict a fault, while  $\langle k, l \rangle$  means that the run is correct so far but a fault can be predicted to happen between the next  $k$  and  $k + l$  observations. When  $l = \omega$ , a fault is predicted but without an upper bound.  $\langle ?, m \rangle$  means that a fault may or may not have happened yet but one will surely occur within  $m$  further observations, and  $\perp$  means that a fault has definitely already occurred.

► **Example 10.** Consider again the LTS from Figure 1 and assume that  $\Sigma_1 = \{a\}$  and  $\Sigma_2 = \{b\}$ . At the beginning, no fault can be predicted, so a controller would be expected to emit the prediction  $\top$ . After observing  $b$ , the controller could predict that a fault will happen between the first and second next observation to come, i.e.  $\langle 1, 1 \rangle$ . After seeing  $d$ , this would change to  $\langle 0, 1 \rangle$ , and finally to  $\perp$ .

► **Definition 11** (predictions). *Let  $\mathbb{P} := \{\top\} \cup (\mathbb{N} \times \mathbb{N}_{\omega}^+) \cup (\{?\} \times \mathbb{N}_{\omega}^+) \cup \{\perp\}$  be the set of possible predictions. We define the following measures  $\kappa, \mu : \mathbb{P} \rightarrow \mathbb{N}_{\omega} \cup \{-1, \omega + 1\}$ :*

- $\kappa(\top) = \omega + 1$ ,  $\kappa(\langle k, l \rangle) = k$ , and  $\kappa(p) = -1$  otherwise;
- $\mu(\top) = \omega + 1$ ,  $\mu(\langle k, l \rangle) = k + l$ ,  $\mu(\langle ?, m \rangle) = m$ , and  $\mu(\perp) = 0$ .

*We also define two particular types of subsets of  $\mathbb{P}$ : For  $k \in \mathbb{N}$  and  $l \in \mathbb{N}^+$ , let  $\mathbb{P}_{k,l} := \{\top, \perp\} \cup \{\langle k', l' \rangle \mid k' \leq k, l' \leq l\} \cup \{\langle ?, m \rangle \mid m < l\}$  and  $\mathbb{P}_{k,\omega} := \{\top, \perp, \langle ?, \omega \rangle\} \cup \{\langle k', \omega \rangle \mid k' \leq k\}$ .*

The values  $\kappa(p)$  and  $\mu(p)$  define the “window” (lower and upper bound on future observations) within which a fault is to occur according to prediction  $p$ . Here,  $-1$  indicates that the fault may or must have occurred in the past, and in the case of  $\top$ ,  $\omega + 1$  is chosen for technical convenience. A predictor using values from  $\mathbb{P}_{k,l}$  makes firm commitments on both the lower and upper bounds within which a fault is going to occur, while a predictor with values from  $\mathbb{P}_{k,\omega}$  only commits to a lower bound.

► **Definition 12** (compatible predictions). *Let  $p, p' \in \mathbb{P}$  and  $k \in \mathbb{N}, l \in \mathbb{N}_{\omega}^+$ . We say that  $\langle p, p' \rangle$  are  $k$ - $l$ -compatible if the following conditions are all satisfied:*

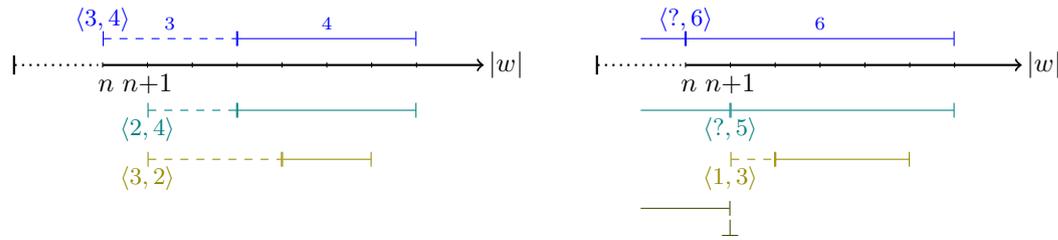
- if  $p = \top$ , then  $\kappa(p') \geq k$  else  $\kappa(p') \geq \kappa(p) - 1$ ;
- $\mu(p') \leq \mu(p)$ , and if  $0 < \mu(p) < \omega$ , then  $\mu(p') < \mu(p)$ ;
- if  $p' \neq \top$ , then  $\mu(p') \leq k + l$ .

*Moreover,  $p$  is called  $k$ - $l$ -initial if  $\langle \top, p \rangle$  are  $k$ - $l$ -compatible.*

The conditions in Definition 12 describe the relations that should reasonably hold between a prediction  $p$  made for some observation  $w$  and the prediction  $p'$  made when one has observed one additional event in a  $k$ - $l$ -predictable controlled LTS. Intuitively these are:

1. When a fault is first predicted, it should be at least  $k$  observations in advance, and the gap between this lower bound and the upper bound should be at most  $l$ . This is why  $p = \top$  should imply  $\kappa(p') \geq k$ . In particular, one cannot switch from  $\top$  to  $\langle k', l' \rangle$  for any  $k' < k$ , nor directly to  $\langle ?, m \rangle$  or  $\perp$ . Moreover, the third condition ensures that when switching from  $\top$  to  $\langle k', l' \rangle$ , we have  $k' + l' \leq k + l$ , which with  $k' \geq k$  implies  $l' \leq l$ .
2. Having predicted a fault within a certain “window”, the subsequent predictions can only become more precise. Thus, one can maintain or shrink that window, but not enlarge, shift, or forget about it. Figure 2 illustrates this idea. E.g., when a predictor announces a fault between the 3rd and 7th following observation, expressed by  $p = \langle 3, 4 \rangle$ , then one step later it must give  $p' = \langle 2, 4 \rangle$  or a more precise verdict such as  $\langle 3, 2 \rangle$ . As another example, if the controller has arrived at a verdict of  $\langle ?, 6 \rangle$ , meaning “a fault has occurred, or will occur within six further observations”, then the information gained from an additional observation may lead it to conclude that the fault has now definitely occurred ( $\perp$ ), will occur later (e.g.,  $\langle 1, 3 \rangle$ ), or to maintain the prediction (e.g.,  $\langle ?, 5 \rangle$ ). Note that  $\langle ?, 6 \rangle$  could only be reached by passing through  $\langle 0, m \rangle$ , for some  $m > 6$ , earlier in the observation. These relations are ensured by allowing  $\kappa$  to decrease by at most one and requiring  $\mu$  to strictly decrease (if an upper bound was given).

A  $k$ - $l$ -initial prediction is one that is admissible for the empty observation.



■ **Figure 2** Examples of compatible predictions  $\langle p, p' \rangle$  after  $n$  resp.  $n + 1$  observations, where  $p$  is illustrated above the timeline, and  $p'$  is one of the predictions below. Solid intervals indicate periods in which a fault is predicted.

► **Definition 13** (active predictor). *Let  $\mathcal{A}$  be an LTS,  $\mathbb{P}' \subseteq \mathbb{P}$ , and  $h = \langle \text{cont}, \text{pred} \rangle$ , where  $\text{cont}$  is a controller and  $\text{pred}$  is a mapping from  $\mathcal{P}(\mathcal{L}^*(\mathcal{A}_{\text{cont}}))$  to  $\mathbb{P}'$ . We call  $h$  a  $k$ - $l$ -active predictor over  $\mathbb{P}'$ , for  $k \in \mathbb{N}$  and  $l \in \mathbb{N}_\omega^+$ , if and only if:*

- (i)  $\mathcal{A}_{\text{cont}}$  is live;
- (ii)  $\text{pred}(\varepsilon)$  is  $k$ - $l$ -initial;
- (iii) for  $w \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}_{\text{cont}}))$ , the prediction satisfies the following:
  - if  $\text{pred}(w) = \top$ , then  $w$  is  $(k + 1)$ -correct in  $\mathcal{A}_{\text{cont}}$ ;
  - if  $\text{pred}(w) = \langle k', l' \rangle$ , then  $w$  is  $k'$ - $l'$ -faulty in  $\mathcal{A}_{\text{cont}}$ ;
  - if  $\text{pred}(w) = \langle ?, m \rangle$ , then  $w$  is ambiguous and  $m$ -faulty in  $\mathcal{A}_{\text{cont}}$ ;
  - if  $\text{pred}(w) = \perp$ , then  $w$  is surely faulty in  $\mathcal{A}_{\text{cont}}$ ;
- (iv) for  $a \in \Sigma_o$ ,  $w, wa \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}_{\text{cont}}))$ ,  $\langle \text{pred}(w), \text{pred}(wa) \rangle$  are  $k$ - $l$ -compatible.

Intuitively, condition (i) requires that the control cannot introduce deadlocks, and conditions (ii),(iii) ensure that the predictions have the intended semantics. Condition (iv) ensures compatibility between two subsequent predictions along an observation. If there exists a  $k$ - $l$ -active predictor for  $\mathcal{A}$ , we call  $\mathcal{A}$   $k$ - $l$ -active-predictable, or just *actively predictable*. Moreover,  $\mathcal{A}$  is called *actively safe* if there exists an active predictor for  $\mathcal{A}$  over  $\{\top\}$ , which entails that  $\mathcal{A}_{\text{cont}}$  is safe.

► **Example 14.** In the LTS  $\mathcal{A}$  of Figure 1, assume that  $\Sigma_1 = \{a, c\}$ ,  $\Sigma_2 = \{a, b\}$ ,  $\Sigma_c = \{a, b, c\}$ . Let  $h = \langle cont, pred \rangle$  be defined by:

- $cont(\varepsilon) = \{b, c, d, f\}$ , and  $cont(w) = \Sigma$  otherwise;
- $pred(\varepsilon) = pred(w) = \top$ , where  $w \in c\Sigma_o^* \cap \mathcal{P}(\mathcal{L}^*(\mathcal{A}))$ ,  $pred(b) = \langle 1, 1 \rangle$ ,  $pred(bd) = \langle 0, 1 \rangle$ , and  $pred(bda^+) = \perp$ .

In this example,  $h$  is a 1-1-active predictor.

Proposition 15 and Proposition 16 will exhibit a tight correspondence between the existence of a  $k$ - $l$ -predictor for  $\mathcal{A}$  and the existence of a controller that makes  $\mathcal{A}$   $k$ - $l$ -predictable. Additionally, Proposition 16 shows that the set of predictions used in a predictor can be limited to a finite set, either committing the prediction to a lower and upper bound for the occurrence of a fault, or just a lower bound.

► **Proposition 15.** If  $h = \langle cont, pred \rangle$  is a  $k$ - $l$ -active predictor for an LTS  $\mathcal{A}$ , then  $\mathcal{A}_{cont}$  is  $k$ - $l$ -predictable.

► **Proposition 16.** Let  $\mathcal{A}$  be an LTS. If there exists a controller  $cont$  such that  $\mathcal{A}_{cont}$  is live and  $k$ - $l$ -predictable, then there exist  $k$ - $l$ -active predictors  $h = \langle cont, pred \rangle$  for  $\mathcal{A}$  over both  $\mathbb{P}_{k,l}$  and  $\mathbb{P}_{k,\omega}$ .

Finally, we introduce the notion of *pilot* as an automata-based representation of  $k$ - $l$ -active predictors. In Section 3 we will show how to find a finite-state pilot when  $\mathcal{A}$  is actively predictable and finite-state.

► **Definition 17 (pilot).** Let  $\mathcal{A}$  be an LTS, then  $\mathcal{C} = \langle \mathcal{B}_C, cont_C, pred_C \rangle$  is called pilot for  $\mathcal{A}$  over  $\mathbb{P}' \subseteq \mathbb{P}$  if  $\mathcal{B}_C = \langle Q^c, q_0^c, \Sigma_o, T^c \rangle$  is a deterministic LTS with labellings  $\langle cont_C, pred_C \rangle : Q^c \rightarrow 2^\Sigma \times \mathbb{P}'$ . Let  $h_C = \langle cont, pred \rangle$  associated with  $\mathcal{C}$  be defined by  $cont(w) = cont_C(q)$  and  $pred(w) = pred_C(q)$  for all  $w \in \mathcal{P}(\mathcal{L}^*(\mathcal{A}))$ , where  $q$  is the unique state such that  $q_0^c \xrightarrow{w} q$ . Then  $\mathcal{C}$  is a  $k$ - $l$ -active predictor for  $\mathcal{A}$  if  $h_C$  is one.

### 3 Controller construction

We solve the decision and synthesis problems simultaneously. We try to construct a pilot-based  $k$ - $l$ -active predictor over some  $\mathbb{P}' \subseteq \mathbb{P}$  for an LTS  $\mathcal{A}$ . The construction succeeds if and only if  $\mathcal{A}$  is  $k$ - $l$ -actively predictable. According to Definition 13, the main challenges in building an active predictor are to ensure that (i) the controlled system remains live, (ii) the fault can be predicted at least  $k$  observations before its occurrences, and (iii) the prediction information is provided.

Our solution consists in building a turn-based game (see [12] for turn-based games) by taking into account the control that has already been performed.

► **Definition 18 (turn-based game).** A game  $\mathcal{G}$  with two players called Control and Environment is a tuple  $\langle V_C, V_E, E, v_0, WIN \rangle$ , where:

- $V_C, V_E$  are the vertices owned by Control and Environment, respectively, and  $V_G = V_C \uplus V_E$  denoting all vertices, with  $v_0 \in V_C$  being an initial vertex;
- $E \subseteq V_G \times V_G$  is a set of directed edges such that for all  $v \in V_G$ , there exists  $(v, v') \in E$ ;
- $WIN \subseteq V_G^\omega$  is a set of winning sequences.

Given a sequence  $\rho = v_0 v_1 \dots v_n$ , we denote  $\rho[i] = v_i$ . A *play* is a sequence of  $V_G^\omega$  such that  $\rho[0] = v_0$  and  $\langle \rho[i], \rho[i+1] \rangle \in E$  for all  $i \geq 0$ ; we call  $\rho^k := \rho[0] \dots \rho[k]$ , for some  $k \geq 0$ , a *partial play* if  $\rho[k] \in V_C$ , and define  $last(\rho^k) := \rho[k]$ . We write  $Play^*(\mathcal{G})$  for the set of partial plays of  $\mathcal{G}$ . A play  $\rho$  is called *winning* (for Control) if  $\rho \in WIN$ .

A Büchi game  $\langle V_C, V_E, E, v_0, V_F \rangle$  defines a game  $\langle V_C, V_E, E, v_0, WIN \rangle$  such that  $WIN = \{\rho \in V_G^\omega \mid \rho[i] \in V_F \text{ for infinitely many } i\}$ . A reachability game  $\langle V_C, V_E, E, v_0, V_F \rangle$  defines a game  $\langle V_C, V_E, E, v_0, WIN \rangle$  such that  $WIN = V_G^* V_F V_G^\omega$ . A safety game  $\langle V_C, V_E, E, v_0, V_F \rangle$  defines a game  $\langle V_C, V_E, E, v_0, WIN \rangle$  such that  $WIN = V_F^\omega$ .

► **Definition 19** (strategy). Let  $\mathcal{G} = \langle V_C, V_E, E, v_0, WIN \rangle$  be a game. A strategy (for Control) is a function  $\theta: \text{Play}^*(\mathcal{G}) \rightarrow V_G$  such that  $(\text{last}(\xi), \theta(\xi)) \in E$  for all  $\xi \in \text{Play}^*(\mathcal{G})$ . A play  $\rho$  adheres to  $\theta$  if  $\rho[i] \in V_C$  implies  $\rho[i+1] = \theta(\rho^i)$  for all  $i \geq 0$ . A strategy is called winning if every play  $\rho$  that adheres to  $\theta$  is winning. A positional (also called memoryless) strategy is a function  $\theta': V_C \rightarrow V_G$  such that  $(v, \theta'(v)) \in E$  for all  $v \in V_C$ ; we call  $\theta'$  winning if the strategy  $\theta$  with  $\theta(\xi) = \theta'(\text{last}(\xi))$  is winning.

To verify  $k$ - $l$ -active predictability of a given system, the controller that we propose needs to memorize two subsets of states with the corresponding prediction information  $\langle Q_c, Q_f, p \rangle$ . The subset  $Q_c$  (resp.  $Q_f$ ) represents the possible states reached by a correct (resp. faulty) run after the last observable action, and  $Q_c \cup Q_f \neq \emptyset$ . The prediction information  $p \in \mathbb{P}$  is (non-deterministically) decided based on the current observations. We denote  $\text{Reach}(\langle Q_c, Q_f, p \rangle) := Q_c \cup Q_f$  and  $\tilde{Q} := 2^Q \setminus \{\emptyset\}$ . The set of possible tuples memorized by the controller is defined as  $S_{\mathbb{P}'} = S_{\mathbb{P}'}^c \cup S_{\mathbb{P}'}^a \cup S_{\mathbb{P}'}^f$ , where:

- $S_{\mathbb{P}'}^c = \tilde{Q} \times \{\emptyset\} \times \{p \in \mathbb{P}' \mid \kappa(p) \geq 0\}$
- $S_{\mathbb{P}'}^a = \tilde{Q} \times \tilde{Q} \times (\mathbb{P}' \cap (\{?\} \times \mathbb{N}_\omega^+))$
- $S_{\mathbb{P}'}^f = \{\emptyset\} \times \tilde{Q} \times \{\perp\}$

In the following, we will simply write  $S$  for  $S_{\mathbb{P}'}$  when  $\mathbb{P}'$  is clear from context.

The controller needs to update the state subsets after an observable action, for which we first define some sets of possible next states from a given state  $q$  after  $a \in \Sigma_o$ .

- $NO_{\mathcal{A}}(q, a) = \{q' \mid q \xrightarrow{\sigma} q', \sigma \in \Sigma_{uo}^* a\}$
- $NOC_{\mathcal{A}}(q, a) = \{q' \mid q \xrightarrow{\sigma} q', \sigma \in (\Sigma_{uo} \setminus \{f\})^* a\}$
- $NOF_{\mathcal{A}}(q, a) = \{q' \mid q \xrightarrow{\sigma} q', \sigma \in \Sigma_{uo}^* f \Sigma_{uo}^* a\}$

One can omit the subscript  $\mathcal{A}$  when there is no ambiguity. The extension to a set of states is defined in a natural way, e.g.  $NO(Q', a) = \bigcup_{q \in Q'} NO(q, a)$ . We now define how the controller updates its tuple once an observable action occurs. In the following,  $\emptyset$  represents a state in which the controller has lost, and we denote  $S^\emptyset := S \cup \{\emptyset\}$ .

► **Definition 20** (knowledge update). Let  $\mathcal{A}$  be an LTS,  $\mathbb{P}' \subseteq \mathbb{P}$ , and  $k \geq 0$ . Then the knowledge transition relation  $\Delta_{\mathcal{A}}^k \subseteq S \times \Sigma_o \times S^\emptyset$  is defined as follows. Let  $s = \langle Q_c, Q_f, p \rangle \in S$  and  $a \in \Sigma_o$ . Then  $\langle s, a, s' \rangle \in \Delta_{\mathcal{A}}^k$  if and only if:

1. either  $s' = \langle NOC(Q_c, a), NOF(Q_c, a) \cup NO(Q_f, a), p' \rangle \in S$  and  $\langle p, p' \rangle$  are  $k$ - $l$ -compatible;
2. or  $s' = \emptyset$  when there is no  $s'' \in S$  such that  $\langle s, a, s'' \rangle \in \Delta_{\mathcal{A}}^k$ .

Notice that, given  $s$  and  $a$ , the choice of  $s'$  is largely deterministic except for  $p'$ , which must be  $k$ - $l$ -compatible with  $p$ . When  $s'$  has no prediction consistent with the updated correct resp. faulty state subsets, cf Definition 13(iii), then the only possible update is to  $\emptyset$ .

► **Example 21.** Consider the LTS in Figure 1 and assume that  $\Sigma_1 = \{a, c\}$ ,  $\Sigma_2 = \{a, b\}$  and  $\Sigma_c = \{a, b, c\}$ .

1. Let  $s = \langle \{q_0\}, \emptyset, \top \rangle$ . If the observable action  $a$  is chosen, then we have  $\langle s, a, s' \rangle \in \Delta_{\mathcal{A}}^k$ , where  $s' = \langle \{q_1, q_4\}, \emptyset, \top \rangle$ . Notice that  $\langle \top, \top \rangle$  are  $k$ - $l$ -compatible.
2. Let  $s = \langle \{q_2, q_5\}, \emptyset, \top \rangle$  after observing  $a$  and  $d$ . If  $a$  is chosen from here, we can only have  $\langle s, a, \emptyset \rangle \in \Delta_{\mathcal{A}}^k$ . The reason is that after  $a$ , the system can end up in either  $q_3$  (with a fault) or in  $q_5$  (without fault), the next prediction should thus be an ambiguous one,

i.e.,  $\langle ?, m \rangle$ . However,  $\langle \top, \langle ?, m \rangle \rangle$  are not  $k$ - $l$ -compatible. It follows that there does not exist  $s'' \in S$  such that  $\langle s, a, s'' \rangle \in \Delta_{\mathcal{A}}^k$ . Hence we have  $\langle s, a, \emptyset \rangle \in \Delta_{\mathcal{A}}^k$  by Definition 20.

The objective of Control is to obtain a winning play by suitably restricting the possible actions, and any winning strategy corresponds to a controller with which the controlled system is predictable. The game begins with Control to choose a prediction for  $\varepsilon$ . Then the game proceeds in rounds: 1) Control restricts the set of possible actions to some  $\Sigma'$ ; 2) Environment chooses  $a \in \Sigma'$  to determine the next state. 3) Control updates its knowledge.

The choices of Control are subject to some restrictions. Indeed, each state  $s = \langle Q_c, Q_f, p \rangle$  represents Control's knowledge about the current potential states of  $\mathcal{A}$  as well as the corresponding prediction information. To ensure that the controlled system remains live, the set of possible actions  $\Sigma'$  must not cause deadlocks in any state reachable by unobservable actions from  $Q_c \cup Q_f$ . Also, Control cannot prevent the uncontrollable actions. So we define the admissible sets and the game as follows, where we use  $\Sigma_{PO}(q) = \{a \in \Sigma_o \mid q \xrightarrow{\sigma} q'', \sigma \in \Sigma_{uo}^* a\}$  to denote the possible next observable actions from the state  $q$ , which can be extended to a set of states in a natural way.

► **Definition 22** (admissible action set). *Let  $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$  be an LTS and  $Q' \subseteq Q$  be a subset of states. We call  $\Sigma' \subseteq \Sigma_o$  an admissible set for  $Q'$  if it fulfills the following conditions:*

- $\Sigma_{uco} \subseteq \Sigma'$  as any action in  $\Sigma_{uco}$  is observable but not controllable.
- for all  $q' \in Q'$ ,  $q \in Q$ , and  $\sigma \in \Sigma_{uo}^*$ ,  $q' \xrightarrow{\sigma} q$  implies  $\Sigma_{PO}(q) \cap \Sigma' \neq \emptyset$ .

*The set of admissible sets for  $Q'$  are denoted as  $\text{adm}(Q')$ , which is not empty when  $Q' \neq \emptyset$  as  $\mathcal{A}$  is a live and convergent LTS.*

► **Example 23.** Consider the same LTS as in Example 21. Let  $Q' = \{q_0\}$ . Then  $\text{adm}(Q') = \{\Sigma' \mid \Sigma' \subseteq \Sigma_o, \{d\} \subsetneq \Sigma'\}$ . In other words,  $\text{adm}(Q')$  contains all subsets of  $\Sigma_o = \{a, b, c, d\}$  that include  $d$ , except the singleton  $\{d\}$ , which is not an admissible set as it blocks the system. More precisely, the set of possible next observable actions from  $q_0$  is  $\Sigma_{PO}(q_0) = \{a, b, c\}$ , whose intersection with  $\{d\}$  is empty. Thus  $\{d\}$  cannot be an admissible set for  $Q'$ .

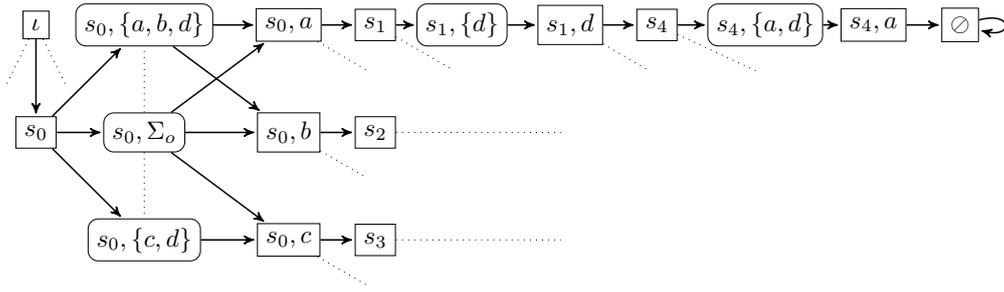
The vertices of our controller-synthesis game consist of an initial vertex  $\iota$ , the states of  $S^\emptyset$ , a set  $V_1 := S \times 2^{\Sigma_o}$  where Control has chosen a set of permitted actions, and a set  $V_2 := S \times \Sigma_o$  where Environment has chosen an observable action. The winning condition assures that once a fault has been predicted, it will eventually happen.

► **Definition 24** (controller-synthesis game). *Let  $\mathcal{A}$  be an LTS and  $\mathbb{P}' \subseteq \mathbb{P}$ . We denote  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k,l}$  the Büchi game  $\langle V_C, V_E, E, \iota, V_F \rangle$ , where  $V_C = \{\iota\} \cup S^\emptyset \cup V_2$ ,  $V_E = V_1$ ,  $V_F = (\tilde{Q} \times \{\emptyset\} \times \{\top\}) \cup (\{\emptyset\} \times \tilde{Q} \times \{\perp\}) \subseteq S$ , and  $E = E_\iota \cup E_1 \cup E_2 \cup E_3 \cup \{\langle \emptyset, \emptyset \rangle\}$ , where*

- $E_\iota = \{\langle \iota, \langle \{q_0\}, \emptyset, p \rangle \rangle \mid p \text{ is } k\text{-}l\text{-initial}\} \subseteq \{\iota\} \times S$ ;
- $E_1 = \{\langle s, \langle s, \Sigma' \rangle \rangle \mid s \in S, \Sigma' \in \text{adm}(\text{Reach}(s))\} \subseteq S \times V_1$ ;
- $E_2 = \{\langle \langle s, \Sigma' \rangle, \langle s, a \rangle \rangle \mid s \in S, a \in \Sigma_{PO}(\text{Reach}(s)) \cap \Sigma'\} \subseteq V_1 \times V_2$ ;
- $E_3 = \{\langle \langle s, a \rangle, s' \rangle \mid \langle s, a, s' \rangle \in \Delta_{\mathcal{A}}^k\} \subseteq V_2 \times S^\emptyset$ .

Note that the set  $V_2$  records the sequence of observable actions that occur during a play.

► **Example 25.** Figure 3 depicts a part of a game for some  $k, l$  and the LTS of Figure 1, for which we assume again  $\Sigma_1 = \{a, c\}$ ,  $\Sigma_2 = \{a, b\}$  and  $\Sigma_c = \{a, b, c\}$ . From  $\iota$ , Controller can choose any  $k$ - $l$ -initial prediction; we consider the case where  $\top$  is chosen, so  $s_0 = \langle \{q_0\}, \emptyset, \top \rangle$ . Then from Example 23, we have  $\text{adm}(\text{Reach}(s_0)) = \text{adm}(\{q_0\}) = \{\Sigma' \mid \Sigma' \subseteq \Sigma_o, \{d\} \subsetneq \Sigma'\}$ . Environment cannot choose the action  $d$  even when  $d$  is in the admissible set since  $d \notin \Sigma_{PO}(\text{Reach}(s_0))$ . After Environment chooses an available action (say  $a$ , leading to  $\langle s_0, a \rangle$ ), Control updates its knowledge and chooses a new prediction, say  $\top$ , leading to  $s_1$ , with  $q_1, q_4$



■ **Figure 3** Part of the game for the LTS in Figure 1 (Example 25):

$s_0 = \langle \{q_0\}, \emptyset, \top \rangle$ ,  $s_1 = \langle \{q_1, q_4\}, \emptyset, \top \rangle$ ,  $s_2 = \langle \{q_1\}, \emptyset, p_2 \rangle$ ,  $s_3 = \langle \{q_4\}, \emptyset, p_3 \rangle$ , and  $s_4 = \langle \{q_2, q_5\}, \emptyset, \top \rangle$ .

as the possible new states. From here,  $d$  is the only choice for Environment. Suppose that Control then again chooses  $\top$  as its new prediction in  $s_4$ , thus  $s_4 = \langle \{q_2, q_5\}, \emptyset, \top \rangle$ . If  $a$  is now chosen, from the second case of Example 21, we know that the game enters  $\emptyset$ . To avoid losing, Control needs to switch to a different prediction early enough.

Now we establish the strong connection between winning strategies and active predictors.

► **Proposition 26.** Given  $h = \langle cont, pred \rangle$  a  $k$ - $l$ -active predictor over  $\mathbb{P}'$  for an LTS  $\mathcal{A}$ , there exists a corresponding winning strategy  $\theta_h$  in the game  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$ .

The existence of a winning strategy implies the existence of a positional one due to well-known results of game theory (see e.g. [12] for all results here related to turn-based games). For the reverse direction, we next define a pilot from a positional winning strategy in  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$  before proving that this pilot is a  $k$ - $l$ -active predictor.

► **Definition 27.** Let  $\theta$  be a positional winning strategy in  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$ . We define a pilot  $\mathcal{C}_\theta := \langle \mathcal{B}_\theta, cont_\theta, pred_\theta \rangle$  over  $\mathbb{P}'$  as follows:

- $\mathcal{B}_\theta = \langle Q^\theta, q_0^\theta, \Sigma_o, T^\theta \rangle$ , where
  1.  $Q^\theta = \{q \in S \mid q = last(\xi_\theta) \text{ and } \xi_\theta \in Play^*(\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}) \text{ adhering to } \theta\}$
  2.  $q_0^\theta = \theta(l)$
  3.  $T^\theta(s, a) = \theta(\langle s, a \rangle)$
- $cont_\theta(s) = \Sigma' \cup \Sigma_{uo}$  for any  $s \in Q^\theta$ , where  $\theta(s) = \langle s, \Sigma' \rangle$ ;
- $pred_\theta(s) = p$ , for any  $s = \langle Q_c, Q_f, p \rangle \in Q^\theta$

► **Proposition 28.** Let  $\theta$  be a positional winning strategy in  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$ . Then  $\mathcal{C}_\theta$  is a  $k$ - $l$ -active predictor over  $\mathbb{P}'$  for  $\mathcal{A}$ .

Combining the results of Propositions 26 and 28, we obtain that the active-predictability problem for an LTS  $\mathcal{A}$  with  $n$  states reduces to solving a Büchi game with  $2^{\mathcal{O}(n)}$  vertices. Since Büchi games can be solved in polynomial time, we obtain the following result:

► **Theorem 29.** The active-predictability problem for finite-state LTS belongs to EXPTIME.

We conclude the section with a supplementary result showing that due to the special structure of  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$  it can actually be solved in linear time (w.r.t. the size of the game), and not in quadratic time as performed for general Büchi games.

► **Proposition 30.** If  $\mathcal{A}$  is a finite-state LTS and  $\mathbb{P}' \subseteq \mathbb{P}$ , then  $\mathcal{G}_{\mathcal{A}, \mathbb{P}'}^{k, l}$  can be solved in  $\mathcal{O}(|E|)$ .

## 4 Bound analysis

We first prove that it is EXPTIME-hard to decide whether a given LTS  $\mathcal{A}$  is actively  $k$ - $l$ -predictable, independently of  $k$  and  $l$ . The proof (developed in [14]) is similar to the proof in [13] that active *diagnosability* is EXPTIME-hard and relies on a reduction from safety games with imperfect information [3].

► **Theorem 31.** *The active-predictability decision problem is EXPTIME-hard.*

Together with Theorem 29, we obtain the following corollary.

► **Corollary 32.** *The active-predictability decision problem is EXPTIME-complete.*

We study the relation between active predictability and active safety. Theorem 33 relates the maximal advance warning for fault predictions to the number of states in  $\mathcal{A}$ .

► **Theorem 33.** *Let  $\mathcal{A}$  be an LTS with  $n$  states. If  $\mathcal{A}$  is  $2^n$ -active-predictable, then it is actively safe.*

**Proof.** If  $\mathcal{A}$  is  $2^n$ - $\omega$ -active-predictable then by definition there exists a  $2^n$ - $\omega$ -active predictor  $h = \langle \text{cont}, \text{pred} \rangle$  over  $\mathbb{P}^l := \mathbb{P}_{k,\omega}$  for  $\mathcal{A}$ , and by Proposition 26 there exists a winning strategy  $\theta$  in  $\mathcal{G}_{\mathcal{A},\mathbb{P}^l}^{k,\omega}$ . In turn, this winning strategy provides a pilot  $\mathcal{C}_\theta = \langle \mathcal{B}, \text{cont}', \text{pred}' \rangle$  according to Proposition 28; let  $\mathcal{B} = \langle Q, q_0, \Sigma_o, T \rangle$ . We shall construct a new pilot  $\mathcal{C}$  for  $\mathcal{A}$  over  $\{\top\}$ , proving that  $\mathcal{A}$  is actively safe.

Remember that  $Q$  is the set of Controller-owned vertices in  $\mathcal{G}_{\mathcal{A},\mathbb{P}^l}^{k,\omega}$ , that can be reached by plays adhering to  $\theta$  and that these vertices are a subset of  $S_{\mathbb{P}^l}$ . For  $q, q' \in Q$ , let us write  $q \prec q'$  if  $q'$  is reachable from  $q$  in  $\mathcal{B}$ . Since  $\theta$  is positional and winning,  $\prec$  must be an acyclic relation between those states of  $Q$  that are not members of  $V_F$ , i.e. their associated prediction is neither  $\top$  nor  $\perp$  (cf Definition 24). We now call  $q \in Q$  a *cutoff* if  $q$  is of the form  $\langle Q_c, Q_f, p \rangle$  and there exists a state  $q' = \langle Q_c, Q_f, p' \rangle$  with  $p' \neq p$  and  $q' \prec q$ . Let  $co(q)$ , the *corresponding state* of  $q$ , denote the state that is  $\prec$ -minimal among all the choices for  $q'$ ; due to the structure of the states outside  $V_F$ ,  $co(q)$  is unique and not a cutoff itself. Moreover, a state of  $Q$  is called *useless* if it is either a cutoff or all its (immediate) predecessors in  $\mathcal{B}$  are useless, and *useful* otherwise.

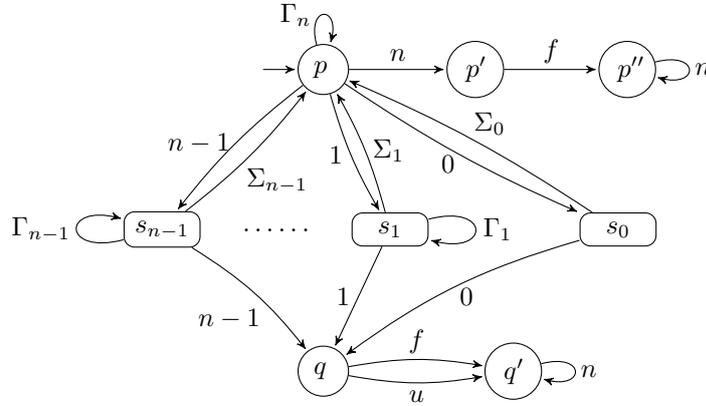
Remember that  $S_{\mathbb{P}^l}$  is a union of  $S_{\mathbb{P}^l}^c$ ,  $S_{\mathbb{P}^l}^a$ , and  $S_{\mathbb{P}^l}^f$ , where  $S_{\mathbb{P}^l}^c$  contains the states of the form  $\langle Q_c, \emptyset, p \rangle$ , with  $\kappa(p) \geq 0$ . Thus, states in  $S_{\mathbb{P}^l}^c$  are only reached through correct runs in  $\mathcal{A}_{\text{cont}'}$ . Let  $S' := \{ \langle Q_c, \emptyset, p \rangle \mid \kappa(p) = 0 \}$ . It follows from the construction of  $\mathcal{G}_{\mathcal{A},\mathbb{P}^l}^{k,\omega}$  (cf Definition 20 and Definition 24) that any path from  $q_0$  to a state from  $S'$  is of length at least  $2^n$ , so by pigeonhole principle, any path leading to  $S'$  contains a cutoff. Since  $S_{\mathbb{P}^l}^a \cup S_{\mathbb{P}^l}^f$  can only be reached by going through  $S'$ , those states are useless.

We can now construct the desired pilot  $\mathcal{C}$  by “folding” cutoffs back onto their corresponding states. We remark in this context that  $\text{Reach}(q) = \text{Reach}(co(q))$ , and therefore the admissible control choices for both states are the same; proving that the resulting controlled system is live depends only on this property. Since the controlled system never admits a fault, the prediction can be  $\top$  in all cases. More formally,  $\mathcal{C} := \langle \langle Q', q_0, \Sigma_o, T' \rangle, \text{cont}', \text{pred}'' \rangle$ , where  $Q'$  is the useful subset of  $Q$ , and for all  $q \in Q'$ ,  $a \in \Sigma_o$ :

- $T'(q, a) = T(q, a)$  if  $T(q, a) \in Q'$  and  $T'(q, a) = co(T(q, a))$  otherwise;
- $\text{pred}''(q) = \top$ . ◀

Theorem 33 implies that if a system is not actively safe, then there is an exponential upper bound on the advance warning that an active predictor can issue. This bound is asymptotically precise, as the following family of examples shows.

► **Theorem 34.** *There exists a family of systems  $(\mathcal{A}_n)_{n \geq 1}$  with  $\mathcal{O}(n)$  states such that  $\mathcal{A}_n$  is not actively safe but  $2^n$ -active-predictable.*



■ **Figure 4** A  $2^n$ -active predictable LTS with  $\mathcal{O}(n)$  states, where  $\Sigma_o = \Sigma_c = \{0, \dots, n\}$ ,  $\Sigma_i = \{i + 1, \dots, n\}$ , and  $\Gamma_i = \{0, \dots, i - 1\}$ .

**Proof.** Figure 4 shows a family of LTS with  $\mathcal{O}(n)$  states but an alphabet of size  $\mathcal{O}(n)$  and  $\mathcal{O}(n^2)$  transitions. We first provide a proof for this family as it is easier to understand. After this, we provide a more complex example with a constant-size alphabet and  $\mathcal{O}(n)$  states and transitions.

### Variable-size alphabet

Consider the LTS shown in Figure 4. The observable actions are  $\{0, \dots, n\}$ , all of which are controllable. There are only two unobservable actions,  $u$  and the fault  $f$ . We abbreviate by  $\Sigma_i := \{i + 1, \dots, n\}$  the actions larger than  $i$  for  $0 \leq i < n$ , and by  $\Gamma_i := \{0, \dots, i - 1\}$  the actions smaller than  $i$  for  $0 < i \leq n$ .

The initial state is  $p$ . Evidently  $\mathcal{A}_n$  is actively safe if a controller can avoid both  $p'$  and  $q$ ; as we shall see, this is impossible. However, the system is actively predictable if the controller can at least avoid  $q$ . We shall see that this is indeed possible while entering  $p'$  only after  $2^n$  steps, by simulating a binary counter.

We can assume (w.l.o.g.) that the controller permits a single action from  $\Sigma_o$  in each step and hence the controlled system will admit a single infinite observation sequence  $\rho$ . Having allowed a prefix  $\sigma$  of  $\rho$ , let  $R(\sigma)$  be the set of states that this sequence can lead to. If the controller wants to keep the system from making a fault, it must ensure that  $R(\sigma)$  remains within the set  $R := \{p, s_0, \dots, s_{n-1}\}$ . When  $R(\sigma) \subseteq R$ , let us associate a measure defined as  $I(\sigma) := \sum_{s_i \in R(\sigma)} 2^i$ . We observe the following:

- $R(\varepsilon) = \{p\}$ , hence  $I(\varepsilon) = 0$ .
- If  $s_i \in R(\sigma)$ , then the controller must not allow action  $i$  in the next step, otherwise the system may go to  $q$ , rendering it unpredictable.
- As long as  $I(\sigma) < 2^n - 1$ , the controller must permit an action  $i$  such that  $I(\sigma i) > I(\sigma)$ . To see this, let  $s_i \notin R(\sigma)$ , then  $R(\sigma i) = (R(\sigma) \cup \{s_i\}) \setminus \{s_0, \dots, s_{i-1}\}$ . We shall assume that  $i$  is chosen minimally, so  $I(\sigma i) = I(\sigma) + 1$ .
- Therefore, after  $2^n - 1$  steps, the controlled system will have performed a sequence  $\hat{\sigma}$  with  $I(\hat{\sigma}) = 2^n - 1$ . The only possible course of action for the controller is to permit  $n$  from now on, i.e.  $\rho = \hat{\sigma}n^\omega$ . We then have  $R(\hat{\sigma}n) = \{p, p'\}$ ,  $R(\hat{\sigma}nn) = \{p', p''\}$ , and  $R(\hat{\sigma}nnn) = \{p''\}$ .

Going backwards, we can now associate predictions with each prefix of  $\rho$ :  $\text{pred}(\hat{\sigma}n^k) = \perp$  for  $k \geq 3$ ,  $\text{pred}(\hat{\sigma}nn) = \langle ?, 1 \rangle$ ,  $\text{pred}(\hat{\sigma}n) = \langle 0, 2 \rangle$ , and  $\text{pred}(\sigma) = \langle 2^n - |\sigma|, 2 \rangle$  for every prefix  $\sigma$  of  $\hat{\sigma}$ . Thus,  $\mathcal{A}_n$  is  $2^n$ -2-active predictable. Notice that the system could be made  $2^n$ -1-active predictable if states  $s_0, \dots, s_{n-1}$  transitioned with  $n$  to  $p'$  instead, which we avoided simply to keep the drawing of the automaton planar.

### Constant-size alphabet

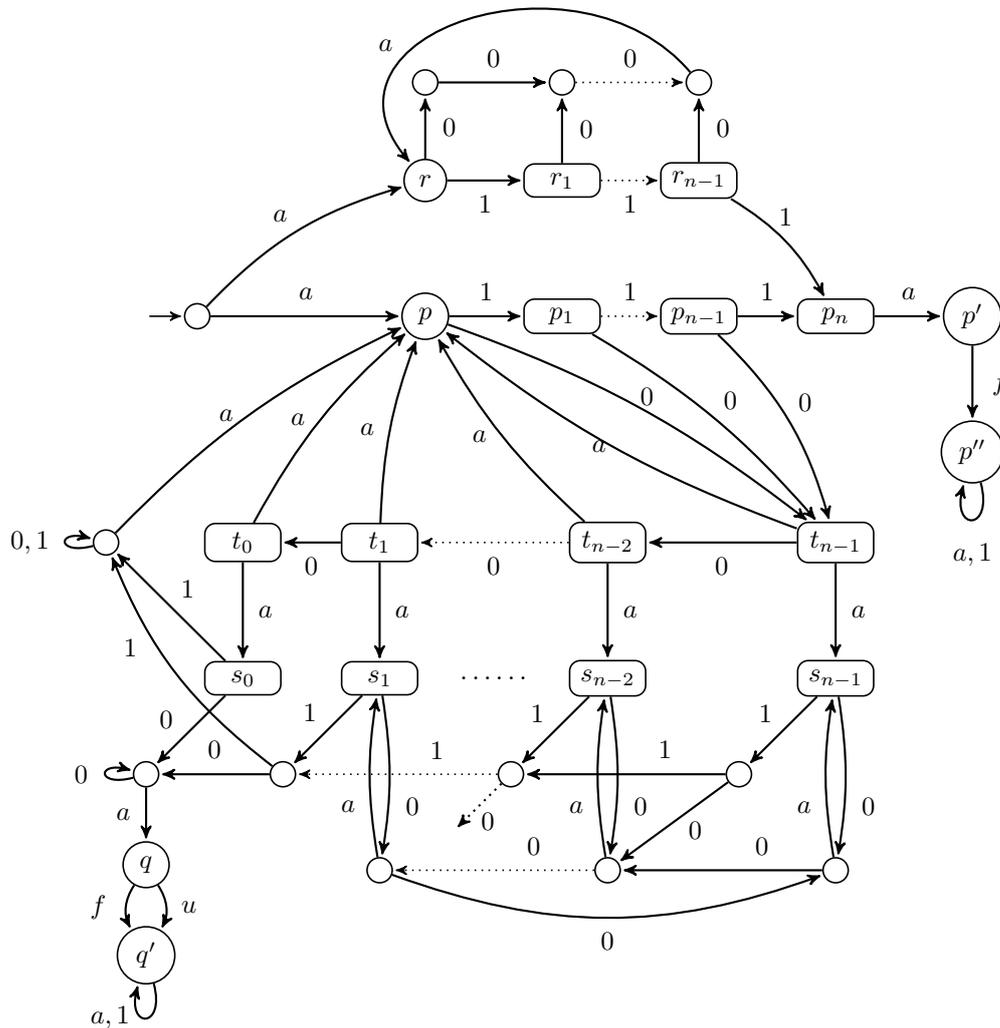
To see that the proof with a variable-size alphabet can be adapted to an alphabet of constant size, consider the LTS  $\mathcal{A}'_n$  in Figure 5.  $\mathcal{A}'_n$  has  $\mathcal{O}(n)$  states and three observable and controllable actions  $0, 1, a$  and two unobservable actions  $u$  and  $f$ . Initially, the LTS performs an  $a$  going to either  $p$  or  $r$ . The LTS then simulates  $\mathcal{A}_n$  of Figure 4, using a unary encoding, in the following sense: Let  $\text{code}(i) = 1^i 0^{n-i} a$ , for  $i = 0, \dots, n$ . The reader can verify, case-by-case, that for any two states  $u, v \in \{p, p', s_0, \dots, s_{n-1}, q\}$  and  $i \in \{0, \dots, n\}$ , we have  $u \xrightarrow{i} v$  in  $\mathcal{A}_n$  iff  $u \xrightarrow{\text{code}(i)} v$  in  $\mathcal{A}'_n$ . Moreover, the controller must account for the possibility that the system has gone to state  $r$ . Then, to keep the controlled system live, the only possible sequences that the controller can enforce are  $\text{code}(i)$  for  $i = 0, \dots, n$ , and we have  $r \xrightarrow{\text{code}(i)} r$  for  $i < n$ . After the initial  $a$ , the controller must therefore admit  $\text{code}(\hat{\sigma}n)$ , for  $\hat{\sigma}$  as in  $\mathcal{A}_n$ . On this basis, a closer look shows that  $\mathcal{A}'_n$  is  $k$ - $l$ -active predictable for  $k = 1 + (n + 1) \cdot 2^n$  and  $l = n + 2$ . ◀

Note that Theorem 34 does not contradict Proposition 8, which establishes linear prediction bounds w.r.t. the number of states of  $\mathcal{A}$ . However, Proposition 8 talks about passive predictability, whereas Theorem 34 is about active predictability.

## 5 Conclusion and perspectives

We have extended the prediction paradigm by introducing parameters related to the number of observations before fault may or must occur. Within this framework, we have established that active predictability is EXPTIME-complete through a procedure for synthesising active predictors that builds a Büchi game. Solving this game is proved linear in the number of edges in the game. We have shown that if the observation threshold for *eventual* prediction is chosen large enough (namely  $\geq 2^n$  with  $n$  the number of states in the system), then active predictability is equivalent to active safety. Furthermore we have exhibited a family of systems proving that this bound is tight.

Out of several possible extensions for the present results, three stand out as natural continuations. First, we want to introduce a measure that quantifies the faultiness of the system, and then aim to find an active predictor that minimizes this criterium, or at least ensures a value below some threshold. Second, we plan to study the notion of prediagnosis introduced in [2] that combines predictability and diagnosability for controllable systems. Finally, we also want to study active predictability for probabilistic systems, as we had previously done for diagnosis in [1].



■ **Figure 5** Variant of Figure 4 with constant-size alphabet, with  $\Sigma_o = \Sigma_c = \{0, 1, a\}$ .

## References

- 1 N. Bertrand, E. Fabre, S. Haar, S. Haddad, and L. Hérouët. Active diagnosis for probabilistic systems. In *FOSSACS 2014, Grenoble, France*, volume 8412 of *LNCS*, pages 29–42, 2014.
- 2 N. Bertrand, S. Haddad, and E. Lefauchaux. Foundation of Diagnosis and Predictability in Probabilistic Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14)*, volume 29 of *LIPICs*, pages 417–429, New Delhi, India, December 2014.
- 3 D. Berwanger and L. Doyen. On the power of imperfect information. In *Proc. FSTTCS*, volume 2 of *LIPICs*, pages 73–82, Bangalore, India, 2008.
- 4 S. Böhm, S. Haar, S. Haddad, P. Hofman, and S. Schwoon. Active diagnosis with observable quiescence. In *Proc. CDC: 54th IEEE Conf. on Decision and Control*, pages 1663–1668, Osaka, Japan, December 2015.

- 5 L. Brandán Briones and A. Madalinski. Bounded predictability for faulty discrete event systems. In *30nd International Conference of the Chilean Computer Science Society, SCCC*, pages 142–146, Curico, Chile, November 2011.
- 6 C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems - Second Edition*. Springer, 2008.
- 7 F. Cassez and S. Tripakis. Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88:497–540, 2008.
- 8 F. Cassez and S. Tripakis. Fault diagnosis with static and dynamic observers. *Fundam. Informaticae*, 88(4):497–540, 2008.
- 9 E. Chantry and Y. Pencolé. Monitoring and active diagnosis for discrete-event systems. In *Proc. SafeProcess'09*, pages 1545–1550, 2009.
- 10 E. Dallal and S. Lafortune. On most permissive observers in dynamic sensor activation problems. *IEEE Trans. Autom. Control.*, 59(4):966–981, 2014.
- 11 S. Genc and S. Lafortune. Predictability of event occurrences in partially-observed discrete-event systems. *Autom.*, 45(2):301–311, 2009. doi:10.1016/j.automat.2008.06.022.
- 12 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
- 13 S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. *Journal of Computer and System Sciences*, 83(1):101–120, 2017.
- 14 Stefan Haar, Serge Haddad, Stefan Schwoon, and Lina Ye. Active Prediction for Discrete Event Systems. working paper or preprint, September 2020. URL: <https://hal.archives-ouvertes.fr/hal-02951944>.
- 15 A. Madalinski and V. Khomenko. Predictability verification with parallel LTL-X model checking based on Petri net unfoldings. *IFAC Proceedings Volumes*, 45(20):1232–1237, 2012. 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes.
- 16 M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, July 1998.
- 17 M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Aut. Cont.*, 40(9):1555–1575, 1995.
- 18 L. Ye, P. Dague, and F. Nouioua. Predictability Analysis of Distributed Discrete Event Systems. In *52nd IEEE Conference on Decision and Control*, pages 5009–5015, Florence, Italy, December 2013.
- 19 X. Yin and S. Lafortune. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Trans. Autom. Control.*, 61(8):2140–2154, 2016.
- 20 X. Yin and S. Lafortune. A general approach for optimizing dynamic sensor activation for discrete event systems. *Autom.*, 105:376–383, 2019.
- 21 X. Yin and Z. Li. Decentralized fault prognosis of discrete event systems with guaranteed performance bound. *Autom.*, 69:375–379, 2016.
- 22 T-S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Trans. Automat. Contr.*, 47(9):1491–1495, 2002.

# Comparing Labelled Markov Decision Processes

Stefan Kiefer 

Department of Computer Science, University of Oxford, UK  
stekie@cs.ox.ac.uk

Qiyi Tang 

Department of Computer Science, University of Oxford, UK  
qiyi.tang@cs.ox.ac.uk

---

## Abstract

A labelled Markov decision process is a labelled Markov chain with nondeterminism, i.e., together with a strategy a labelled MDP induces a labelled Markov chain. The model is related to interval Markov chains. Motivated by applications of equivalence checking for the verification of anonymity, we study the algorithmic comparison of two labelled MDPs, in particular, whether there exist strategies such that the MDPs become equivalent/inequivalent, both in terms of trace equivalence and in terms of probabilistic bisimilarity. We provide the first polynomial-time algorithms for computing memoryless strategies to make the two labelled MDPs inequivalent if such strategies exist. We also study the computational complexity of qualitative problems about making the total variation distance and the probabilistic bisimilarity distance less than one or equal to one.

**2012 ACM Subject Classification** Theory of computation → Program verification; Theory of computation → Models of computation; Mathematics of computing → Probability and statistics

**Keywords and phrases** Markov decision processes, Markov chains, Behavioural metrics

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.49

**Related Version** A full version of the paper is [24], available at <https://arxiv.org/abs/2009.11643>.

**Funding** *Stefan Kiefer*: Supported by a Royal Society University Fellowship.

**Acknowledgements** We thank the anonymous reviewers of this paper for their constructive feedback.

## 1 Introduction

Given a model of computation (e.g., finite automata), and two instances of it, are they semantically equivalent (i.e., do they accept the same language)? Such *equivalence* problems can be viewed as a fundamental question for almost any model of computation. As such, they permeate computer science, in particular, theoretical computer science.

In *labelled Markov chains (LMCs)*, which are Markov chains whose states (or, equivalently, transitions) are labelled with an observable letter, there are two natural and very well-studied versions of equivalence, namely *trace (or language) equivalence* and *probabilistic bisimilarity*.

The *trace equivalence* problem has a long history, going back to Schützenberger [33] and Paz [29] who studied *weighted* and *probabilistic* automata, respectively. Those models generalize LMCs, but the respective equivalence problems are essentially the same. It can be extracted from [33] that equivalence is decidable in polynomial time, using a technique based on linear algebra. Variants of this technique were developed in [38, 16]. More recently, the efficient decidability of the equivalence problem was exploited, both theoretically and practically, for the verification of probabilistic systems, see, e.g., [22, 23, 30, 28, 27]. In those works, equivalence naturally expresses properties such as obliviousness and anonymity, which are difficult to formalize in temporal logic. In a similar vein, inequivalence can mean detectibility and the lack of anonymity.



© Stefan Kiefer and Qiyi Tang;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 49; pp. 49:1–49:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

*Probabilistic bisimilarity* is an equivalence that was introduced by Larsen and Skou [26]. It is finer than trace equivalence, i.e., probabilistic bisimilarity implies trace equivalence. A similar notion for Markov chains, called *lumpability*, can be traced back at least to the classical text by Kemeny and Snell [21]. Probabilistic bisimilarity can also be computed in polynomial time [2, 13, 39]. Indeed, in practice, computing the bisimilarity quotient is fast and has become a backbone for highly efficient tools for probabilistic verification such as PRISM [25] and STORM [19].

In this paper, we study equivalence problems for (*labelled*) *Markov decision processes* (*MDPs*), which are LMCs plus nondeterminism, i.e., each state may have several *actions* (or “moves”) one of which is chosen by a controller, potentially randomly. An MDP and a controller *strategy* together induce an LMC (potentially with infinite state space, depending on the complexity of the strategy). The nondeterminism in MDPs gives rise to a spectrum of equivalence queries: one may ask about the existence of strategies for two given MDPs such that the induced LMCs become trace/bisimulation equivalent, or such that they become trace/bisimulation *inequivalent*. Another potential dimension of this spectrum is whether to consider general strategies or more restricted ones, such as memoryless or even memoryless deterministic (MD) ones.

In this paper, we focus on *memoryless* strategies, for several reasons. First, these questions for unrestricted strategies quickly lead to undecidability. For example, in [17, Theorem 3.1] it was shown that whether there exists a general strategy such that a given MDP becomes trace equivalent with a given LMC is undecidable. Second, memoryless strategies are sufficient for a wide range of objectives in MDPs, and their simplicity means that even if it was known that a general strategy exists to accomplish (in)equivalence one might still wonder if there *also* exists a memoryless strategy. Third, probabilistic bisimilarity is a less natural notion for LMCs induced by general strategies: such LMCs will in general have an infinite state space, even when the MDP is finite. Fourth, applying a memoryless strategy in an MDP is related to choosing an instance of an *interval Markov chain* (*IMC*). IMCs are like Markov chains, but the transitions are labelled not with probabilities but with probability intervals. IMCs were introduced by Jonsson and Larsen [20] and have been well studied in verification-related domains [34, 7, 12, 3, 6], but also in areas such as systems biology, security or communication protocols, see, e.g., [11]. Selecting a memoryless strategy in an MDP corresponds to selecting a probability from each interval (one out of generally uncountably many). *Parametric Markov chains* and *parametric MDPs* are further related models, see, e.g., [18, 41] and the references therein.

LMCs can also be compared in terms of their *distance*. We consider two natural distance functions between two LMCs: the *total variation* distance (between the two trace distributions) and the *probabilistic bisimilarity* distance [15]. Both distances can be at most 1. The total variation (resp. probabilistic bisimilarity) distance is 0 if and only if the LMCs are trace equivalent (resp. probabilistic bisimilar). Further, the probabilistic bisimilarity distance is an upper bound on the total variation distance [8]. It was shown in [9] (resp. [37]) that whether the total variation (resp. probabilistic bisimilarity) distance of two LMCs equals 1 can be decided in polynomial time. This raises the question whether these results can be extended to MDPs, i.e., what is the complexity of deciding whether there exists a memoryless strategy to make the distance less than 1 or equal to 1, respectively. It turns out that some of these problems are closely related to the corresponding (in)equivalence problem.

Instead of comparing two MDPs with initial distributions/states, one may equivalently compare two initial distributions/states in a single MDP (by taking a disjoint union of the states). In this paper we study the computational complexity of the following problems:

- $TV = 0$  ( $TV > 0$ ), which asks whether there is a memoryless strategy such that the two initial distributions are (not) trace equivalent in the induced labelled Markov chain;
- $TV = 1$  ( $TV < 1$ ), which asks whether there is a memoryless strategy such that the two initial distributions (do not) have total variation distance one;
- $PB = 0$  ( $PB > 0$ ), which asks whether there is a memoryless strategy such that the two initial states are (not) probabilistic bisimilar;
- $PB = 1$  ( $PB < 1$ ), which asks whether there is a memoryless strategy such that the two initial states (do not) have probabilistic bisimilarity distance one.

In Sections 3 and 4 we provide the first polynomial-time algorithms for  $TV > 0$  and  $PB > 0$ , respectively. We also show how to compute memoryless strategies that witness trace and probabilistic bisimulation inequivalence, respectively. In Section 5 we discuss  $TV = 1$  and  $PB = 1$ , and in Section 6 we establish the complexity of the remaining four problems, which are about making the distance small ( $= 0$  or  $< 1$ ). We conclude in Section 7. Table 1 summarises the results in the paper. Missing proofs can be found in the full version of this paper [24].

■ **Table 1** Summary of the results. These results also imply results for the problems which state “for all memoryless strategies”. For example,  $TV > 0$  is the complement of the decision problem whether for all memoryless strategies the two initial distributions are trace equivalent in the induced labelled Markov chains.

Problem	Complexity
$TV = 0$	$\exists\mathbb{R}$ -complete
$TV > 0$	in P
$TV = 1$	NP-hard and in $\exists\mathbb{R}$
$TV < 1$	$\exists\mathbb{R}$ -complete
$PB = 0$	NP-complete
$PB > 0$	in P
$PB = 1$	NP-complete
$PB < 1$	NP-complete

## 2 Preliminaries

We write  $\mathbb{R}$  for the set of real numbers and  $\mathbb{N}$  the set of nonnegative integers. Let  $S$  be a finite set. We denote by  $\text{Distr}(S)$  the set of probability distributions on  $S$ . By default we view vectors, i.e., elements of  $\mathbb{R}^S$ , as row vectors. For a vector  $\mu \in [0, 1]^S$  we write  $|\mu| := \sum_{s \in S} \mu(s)$  for its  $L_1$ -norm. A vector  $\mu \in [0, 1]^S$  is a distribution (resp. subdistribution) over  $S$  if  $|\mu| = 1$  (resp.  $0 < |\mu| \leq 1$ ). We denote column vectors by boldface letters; in particular,  $\mathbf{1} \in \{1\}^S$  and  $\mathbf{0} \in \{0\}^S$  are column vectors all whose entries are 1 and 0, respectively. For  $s \in S$  we write  $\delta_s$  for the (Dirac) distribution over  $S$  with  $\delta_s(s) = 1$  and  $\delta_s(r) = 0$  for  $r \in S \setminus \{s\}$ . For a (sub)distribution  $\mu$  we write  $\text{support}(\mu) = \{s \in S \mid \mu(s) > 0\}$  for its support.

A *labelled Markov chain* (LMC) is a quadruple  $\langle S, L, \tau, \ell \rangle$  consisting of a nonempty finite set  $S$  of states, a nonempty finite set  $L$  of labels, a transition function  $\tau : S \rightarrow \text{Distr}(S)$ , and a labelling function  $\ell : S \rightarrow L$ .

We denote by  $\tau(s)(t)$  the transition probability from  $s$  to  $t$ . Similarly, we denote by  $\tau(s)(E) = \sum_{t \in E} \tau(s)(t)$  the transition probability from  $s$  to  $E \subseteq S$ . A trace in a LMC  $\mathcal{M}$  is a sequence of labels  $w = a_1 a_2 \cdots a_n$  where  $a_i \in L$ . We denote by  $L^{\leq n}$  the set of traces of length at most  $n$ . Let  $M : L \rightarrow [0, 1]^{S \times S}$  specify the transitions, so that  $\sum_{a \in L} M(a)$

## 49:4 Comparing Labelled Markov Decision Processes

is a stochastic matrix,  $M(a)(s, t) = \tau(s)(t)$  if  $\ell(s) = a$  and  $M(a)(s, t) = 0$  otherwise. We extend  $M$  to the mapping  $M : L^* \rightarrow [0, 1]^{S \times S}$  with  $M(w) = M(a_1) \cdots M(a_n)$  for a trace  $w = a_1 \cdots a_n$ . If the LMC is in state  $s$ , then with probability  $M(w)(s, s')$  it emits a trace  $w$  and moves to state  $s'$  in  $|w|$  steps. For a trace  $w \in L^*$ , we define  $Run(w) := \{w\}L^*$ ; i.e.,  $Run(w)$  is the set of traces starting with  $w$ . To an initial distribution  $\pi$  on  $S$ , we associate the probability space  $(L^\omega, \mathcal{F}, \Pr_{\mathcal{M}, \pi})$ , where  $\mathcal{F}$  is the  $\sigma$ -field generated by all basic cylinders  $Run(w)$  with  $w \in L^*$  and  $\Pr_{\mathcal{M}, \pi} : \mathcal{F} \rightarrow [0, 1]$  is the unique probability measure such that  $\Pr_{\mathcal{M}, \pi}(Run(w)) = |\pi M(w)|$ . We generalize the definition of  $\Pr_{\mathcal{M}, \pi}$  to subdistributions  $\pi$  in the obvious way, yielding sub-probability measures. We may drop the subscript  $\mathcal{M}$  if it is clear from the context.

Given two initial distributions  $\mu$  and  $\nu$ , the *total variation distance* between  $\mu$  and  $\nu$  is defined as follows:

$$d_{tv}(\mu, \nu) = \sup_{E \in \mathcal{F}} |\Pr_\mu(E) - \Pr_\nu(E)|.$$

We write  $\mu \equiv \nu$  to denote that  $\mu$  and  $\nu$  are trace equivalent, i.e.,  $|\Pr_\mu(Run(w))| = |\Pr_\nu(Run(w))|$  holds for all  $w \in L^*$ . We have that trace equivalence and the total variation distance being zero are equivalent [9, Proposition 3(a)].

The pseudometric *probabilistic bisimilarity distance* of Desharnais et al. [14], which we denote by  $d_{pb}$ , is a function from  $S \times S$  to  $[0, 1]$ , that is, an element of  $[0, 1]^{S \times S}$ . It can be defined as the least fixed point of the following function:

$$\Delta(d)(s, t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \min_{\omega \in \Omega(\tau(s), \tau(t))} \sum_{u, v \in S} \omega(u, v) d(u, v) & \text{otherwise} \end{cases}$$

where the set  $\Omega(\mu, \nu)$  of *couplings* of  $\mu, \nu \in \text{Distr}(S)$  is defined as  $\Omega(\mu, \nu) = \{\omega \in \text{Distr}(S \times S) \mid \sum_{t \in S} \omega(s, t) = \mu(s) \wedge \sum_{s \in S} \omega(s, t) = \nu(t)\}$ . Note that a coupling  $\omega \in \Omega$  is a joint probability distribution with marginals  $\mu$  and  $\nu$  (see, e.g., [4, page 260-262]).

An equivalence relation  $R \subseteq S \times S$  is a *probabilistic bisimulation* if for all  $(s, t) \in R$ ,  $\ell(s) = \ell(t)$  and  $\tau(s)(E) = \tau(t)(E)$  for each  $R$ -equivalence class  $E$ . *Probabilistic bisimilarity*, denoted by  $\sim_{\mathcal{M}}$  (or  $\sim$  when  $\mathcal{M}$  is clear), is the largest probabilistic bisimulation. For all  $s, t \in S$ ,  $s \sim t$  if and only if  $d_{pb}(s, t) = 0$  [14, Theorem 1].

A (*labelled*) *Markov decision process* (MDP) is a tuple  $\langle S, \mathcal{A}, L, \varphi, \ell \rangle$  consisting of a finite set  $S$  of states, a finite set  $\mathcal{A}$  of actions, a finite set  $L$  of labels, a partial function  $\varphi : S \times \mathcal{A} \rightarrow \text{Distr}(S)$  denoting the probabilistic transition, and a labelling function  $\ell : S \rightarrow L$ . The set of available actions in a state  $s$  is  $\mathcal{A}(s) = \{m \in \mathcal{A} \mid \varphi(s, m) \text{ is defined}\}$ . A *memoryless* strategy for an MDP is a function  $\alpha : S \rightarrow \text{Distr}(\mathcal{A})$  that given a state  $s$ , returns a probability distribution on all the available actions at that state. Such strategies are also known as positional, as they do not depend on the history of past states. A strategy  $\alpha$  is *memoryless deterministic* (MD) if for all states  $s$  there exists an action  $m \in \mathcal{A}(s)$  such that  $\alpha(s)(m) = 1$ ; we thus view an MD strategy as a function  $\alpha : S \rightarrow \mathcal{A}$ .

For the remainder of the paper, we fix an MDP  $\mathcal{D} = \langle S, \mathcal{A}, L, \varphi, \ell \rangle$ . Given a memoryless strategy  $\alpha$  for  $\mathcal{D}$ , an LMC  $\mathcal{D}(\alpha) = \langle S, L, \tau, \ell \rangle$  is induced, where  $\tau(s)(t) = \sum_{m \in \mathcal{A}(s)} \alpha(s)(m) \cdot \varphi(s, m)(t)$ . The matrix  $M_\alpha$  specifies the transitions of the LMC  $\mathcal{D}(\alpha)$  as is defined previously.

We fix two initial distributions  $\mu$  and  $\nu$  on  $S$  (resp. two initial states  $s$  and  $t$ ) for problems related to total variation distance (resp. probabilistic bisimilarity distance).

### 3 Trace Inequivalence

In this section we show that one can decide in polynomial time whether there exists a memoryless strategy  $\alpha$  so that  $\mu \not\equiv \nu$  in  $\mathcal{D}(\alpha)$ . In terms of the notation from the introduction, we show that  $\text{TV} > 0$  is in  $\text{P}$ . Define the following column-vector spaces.

$$\begin{aligned}\mathcal{V}_1 &= \langle M_{\alpha_1}(a_1)M_{\alpha_2}(a_2)\cdots M_{\alpha_m}(a_m)\mathbf{1} : \alpha_i \text{ is a memoryless strategy; } a_i \in L \rangle \text{ and} \\ \mathcal{V}_2 &= \langle M_\alpha(w)\mathbf{1} : \alpha \text{ is a memoryless strategy; } w \in L^* \rangle \text{ and} \\ \mathcal{V}_3 &= \langle M_\alpha(w)\mathbf{1} : \alpha \text{ is an MD strategy; } w \in L^* \rangle.\end{aligned}$$

Here and later we use the notation  $\langle \cdot \rangle$  to denote the span of (i.e., the vector space spanned by) a set of vectors. By the definitions, we have that  $\mu \equiv \nu$  in all LMCs induced by all memoryless strategies  $\alpha$  if and only if  $\mu M_\alpha(w)\mathbf{1} = \nu M_\alpha(w)\mathbf{1}$  holds for all memoryless strategies  $\alpha$  and all  $w \in L^*$ . It follows:

► **Proposition 1.** *For all distributions  $\mu, \nu$  over  $S$  we have:*

$$\exists \text{ a memoryless strategy } \alpha \text{ such that } \mu \not\equiv \nu \text{ in } \mathcal{D}(\alpha) \iff \mu \mathbf{v} \neq \nu \mathbf{v} \text{ for some } \mathbf{v} \in \mathcal{V}_2.$$

To decide  $\text{TV} > 0$  and to compute the “witness” memoryless strategy such that  $\mu \not\equiv \nu$  in the induced LMC, it suffices to compute a basis for  $\mathcal{V}_2$ ; more precisely, a set of  $\alpha$  and  $w$  such that the vectors  $M_\alpha(w)\mathbf{1}$  span  $\mathcal{V}_2$ . As the set of memoryless strategies is uncountable, this is not straightforward. From the definitions, we know  $\mathcal{V}_3 \subseteq \mathcal{V}_2 \subseteq \mathcal{V}_1$ . We will show  $\mathcal{V}_1 \subseteq \mathcal{V}_3$  and thus establish the equality of these three vector spaces. It follows from [17, Theorem 5.12] that computing a basis for  $\mathcal{V}_1$  is in  $\text{P}$ . It follows that our problem  $\text{TV} > 0$  is also in  $\text{P}$ , but this does not explicitly give the witnessing memoryless strategy. Since  $\mathcal{V}_2 = \mathcal{V}_3$ , there must exist an MD strategy that witnesses  $\mu \not\equiv \nu$ . To find this MD strategy, one can go through all MD strategies (potentially exponentially many). In the following, by considering the vector spaces while restricting the word length, we show that a witness MD strategy can also be computed in polynomial time.

We define the following column-vector spaces. For each  $j \in \mathbb{N}$ ,

$$\begin{aligned}\mathcal{V}_1^j &= \langle M_{\alpha_1}(a_1)M_{\alpha_2}(a_2)\cdots M_{\alpha_k}(a_k)\mathbf{1} : \alpha_i \text{ is a memoryless strategy; } a_i \in L; k \leq j \rangle \text{ and} \\ \mathcal{V}_2^j &= \langle M_\alpha(w)\mathbf{1} : \alpha \text{ is a memoryless strategy; } w \in L^{\leq j} \rangle \text{ and} \\ \mathcal{V}_3^j &= \langle M_\alpha(w)\mathbf{1} : \alpha \text{ is an MD strategy; } w \in L^{\leq j} \rangle.\end{aligned}$$

Let  $\alpha$  be an MD strategy and  $\mathbf{m}$  be an action available at state  $i$ . Recall that an MD strategy can be viewed as a function  $\alpha : S \rightarrow \mathcal{A}$ . We define  $\alpha^{i \rightarrow \mathbf{m}}$  to be the MD strategy such that  $\alpha^{i \rightarrow \mathbf{m}}(i) = \mathbf{m}$  and  $\alpha^{i \rightarrow \mathbf{m}}(s) = \alpha(s)$  for all  $s \in S \setminus \{i\}$ . Let  $\mathbf{c}_i \in \{0, 1\}^S$  be the column bit vector whose only non-zero entry is the  $i$ th one. For a set  $B \subseteq \mathbb{R}^S$ , we define  $\langle B \rangle$  to be the vector space spanned by  $B$ .

We call a column vector an *MD vector* if it is of the form  $M_\alpha(w)\mathbf{1}$  for an MD strategy  $\alpha$  and  $w \in L^*$ . Let  $P$  be a set of MD strategy and word pairs, i.e.,  $P = \{(\alpha_1, w_1), (\alpha_2, w_2), \dots, (\alpha_m, w_m)\}$  where  $\alpha_i$  is an MD strategy and  $w_i \in L^*$ . We define a function  $\mathcal{B}$  transforming such a set  $P$  to the set of corresponding MD vectors, i.e.,  $\mathcal{B}(P) = \{M_{\alpha_1}(w_1)\mathbf{1}, M_{\alpha_2}(w_2)\mathbf{1}, \dots, M_{\alpha_m}(w_m)\mathbf{1}\}$ .

► **Lemma 2.** *Let  $j \in \mathbb{N}$ . For all MD strategies  $\alpha_1$  and  $\alpha_2$ ,  $a \in L$  and  $w \in L^{\leq j}$ , we have  $M_{\alpha_1}(a)M_{\alpha_2}(w)\mathbf{1} \in \langle \mathcal{V}_1^j \cup \mathcal{B}(\{(\alpha, aw)\}) \rangle$  where  $\alpha$  is the MD strategy defined by*

$$\alpha(i) = \begin{cases} \alpha_1(i) & \text{if } \mathbf{c}_i \notin \mathcal{V}_1^j \\ \alpha_2(i) & \text{otherwise} \end{cases}$$

## 49:6 Comparing Labelled Markov Decision Processes

The next lemma shows that a basis for  $\mathcal{V}_1^j$  for some  $j < |S|$  consisting only of MD vectors can be computed in polynomial time.

► **Lemma 3.** *Let  $j \in \mathbb{N}$  with  $j < |S|$ . One can compute in polynomial time a set  $P_j = \{(\alpha_0, w_0), \dots, (\alpha_k, w_k)\}$  in which all  $\alpha_i$  are MD strategies and all  $w_i$  are in  $L^{\leq j}$  such that  $\mathcal{B}(P_j)$  is a basis of  $\mathcal{V}_1^j$ .*

**Proof sketch.** We prove this lemma by induction on  $j$ . The base case where  $j = 0$  is vacuously true with  $P_0 = \{(\alpha_0, w_0)\}$  where  $\alpha_0$  is an arbitrary MD strategy,  $w_0 = \varepsilon$  and  $\mathcal{B}(P_0) = \{\mathbf{1}\}$ . For the induction step, assume that we can compute in polynomial time a set  $P_j = \{(\alpha_0, w_0), \dots, (\alpha_k, w_k)\}$  where all the strategies are MD strategies and all the words are in  $L^{\leq j}$  such that  $\mathcal{B}(P_j)$  is a basis for  $\mathcal{V}_1^j$ . We show that the statement holds for  $j + 1$ . Define

$$\Sigma = \{\alpha_0\} \cup \{\alpha_0^{s \rightarrow m} : s \in S, m \in \mathcal{A}(s)\} \quad \text{and} \quad \mathbb{M} = \{M_\alpha(a) \in \mathbb{R}^{S \times S} : \alpha \in \Sigma, a \in L\}.$$

Next, we present Algorithm 1 which computes a set  $P_{j+1}$  in polynomial time such that

$$\text{for all } M \in \mathbb{M} \text{ and all } \mathbf{b} \in \mathcal{B}(P_j) : M \cdot \mathbf{b} \in \langle \mathcal{B}(P_{j+1}) \rangle \quad (1)$$

■ **Algorithm 1** Polynomial-time algorithm computing  $P_{j+1}$ .

---

```

1  $P_{j+1} := P_j$ 
2 foreach  $\alpha_1 \in \Sigma, a \in L$  and  $(\alpha_2, w) \in P_j$  do
3   if  $M_{\alpha_1}(a)M_{\alpha_2}(w)\mathbf{1} \notin \langle \mathcal{B}(P_{j+1}) \rangle$  then
4     add  $(\alpha, aw)$  to  $P_{j+1}$  where  $\alpha$  is the MD strategy defined as
           
$$\alpha(i) = \begin{cases} \alpha_1(i) & \text{if } \mathbf{c}_i \notin \mathcal{V}_1^j \\ \alpha_2(i) & \text{otherwise.} \end{cases}$$

5   end
6 end

```

---

All the vectors in  $\mathcal{B}(P_{j+1})$  are linearly independent, as we only add a pair if the corresponding vector is linearly independent to the existing vectors in  $\mathcal{B}(P_{j+1})$  (lines 3-4). Since  $\mathcal{B}(P_j)$  is a basis for  $\mathcal{V}_1^j$ , we can decide whether  $\mathbf{c}_i \in \mathcal{V}_1^j$  for  $i \in S$  in polynomial time, and thus compute a pair  $(\alpha, aw)$  on line 4 in polynomial time. Since  $|\Sigma|$  and  $|L|$  are polynomial in the size of the MDP,  $|P_j| < |S|$ , the number of iterations is polynomial in the size of the MDP. The construction of  $P_{j+1}$  is then in polynomial time. It remains to show that after adding  $(\alpha, aw)$  to  $P_{j+1}$  (line 4), we have  $M \cdot \mathbf{b} = M_{\alpha_1}(a)M_{\alpha_2}(w)\mathbf{1} \in \langle \mathcal{B}(P_{j+1}) \rangle$ . Since the pair  $(\alpha_2, w)$  is in  $P_j$ , we have  $w \in L^{\leq j}$ . Then,

$$\begin{aligned} & M \cdot \mathbf{b} \\ &= M_{\alpha_1}(a)M_{\alpha_2}(w)\mathbf{1} \\ &\in \langle \mathcal{V}_1^j \cup \mathcal{B}(\{(\alpha, aw)\}) \rangle \quad [\text{Lemma 2}] \\ &= \langle \mathcal{B}(P_j) \cup \mathcal{B}(\{(\alpha, aw)\}) \rangle \quad [\mathcal{B}(P_j) \text{ is a basis for } \mathcal{V}_1^j \text{ by induction hypothesis}] \\ &= \langle \mathcal{B}(P_j \cup \{(\alpha, aw)\}) \rangle \end{aligned}$$

Since  $P_j \subseteq P_{j+1}$  (line 1), we have  $\mathcal{B}(P_j) \subseteq \mathcal{B}(P_{j+1})$ . By adding the pair  $(\alpha, aw)$  to  $P_{j+1}$ , we have  $\langle \mathcal{B}(P_j \cup \{(\alpha, aw)\}) \rangle \subseteq \langle \mathcal{B}(P_{j+1}) \rangle$ , and thus  $M \cdot \mathbf{b} \in \langle \mathcal{B}(P_{j+1}) \rangle$ .

Finally, we show that the set  $P_{j+1}$  satisfies  $\mathcal{V}_1^{j+1} = \langle \mathcal{B}(P_{j+1}) \rangle$ . We have

$$\begin{aligned} \langle \mathcal{B}(P_{j+1}) \rangle &\subseteq \mathcal{V}_3^{j+1} && \text{for all } (\alpha, w) \in P_{j+1} : \alpha \text{ is an MD strategy and } w \in L^{\leq j+1} \\ &\subseteq \mathcal{V}_1^{j+1} && \text{from the definitions} \end{aligned}$$

We prove the other direction  $\mathcal{V}_1^{j+1} \subseteq \langle \mathcal{B}(P_{j+1}) \rangle$  in [24].  $\blacktriangleleft$

Combining classical linear algebra arguments about equivalence checking (see, e.g., [38]) with Lemma 3, we obtain:

► **Lemma 4.**

1. For all  $j < |S|$  we have  $\mathcal{V}_1^j = \mathcal{V}_2^j = \mathcal{V}_3^j$ .
2. We have  $\mathcal{V}_1 = \mathcal{V}_2 = \mathcal{V}_3 = \mathcal{V}_1^{|S|-1} = \mathcal{V}_2^{|S|-1} = \mathcal{V}_3^{|S|-1}$ .

Thus we obtain:

► **Proposition 5.** One can compute in polynomial time a set  $P = \{(\alpha_0, w_0), \dots, (\alpha_k, w_k)\}$  of MD strategy and word pairs such that  $\mathcal{B}(P)$  is a basis of  $\mathcal{V}_2$ .

**Proof.** By Lemma 4 it suffices to invoke Lemma 3 for  $j = |S| - 1$ .  $\blacktriangleleft$

Now we can prove the main theorem of this section.

► **Theorem 6.** The problem  $\text{TV} > 0$  is in P. Further, for any positive instance of the problem  $\text{TV} > 0$ , we can compute in polynomial time an MD strategy  $\alpha$  and a word  $w$  that witness  $\mu \not\equiv \nu$ , i.e.,  $\Pr_{\mu, \mathcal{D}(\alpha)}(\text{Run}(w)) \neq \Pr_{\nu, \mathcal{D}(\alpha)}(\text{Run}(w))$ .

**Proof.** A polynomial algorithm follows naturally from Proposition 5 and Proposition 1. We first compute a set  $P$  of MD strategy and word pairs such that  $\mathcal{B}(P)$  is a basis for  $\mathcal{V}_2$ . For each  $\mathbf{b} \in \mathcal{B}(P)$ , we check whether  $\mu \mathbf{b} \neq \nu \mathbf{b}$  and output “yes” indicating a positive instance if the inequality holds. Otherwise, we have  $\mu \mathbf{b} = \nu \mathbf{b}$  for all  $\mathbf{b} \in \mathcal{B}(P)$ , and the algorithm outputs “no” indicating that  $\mu \equiv \nu$  holds for all memoryless strategies.

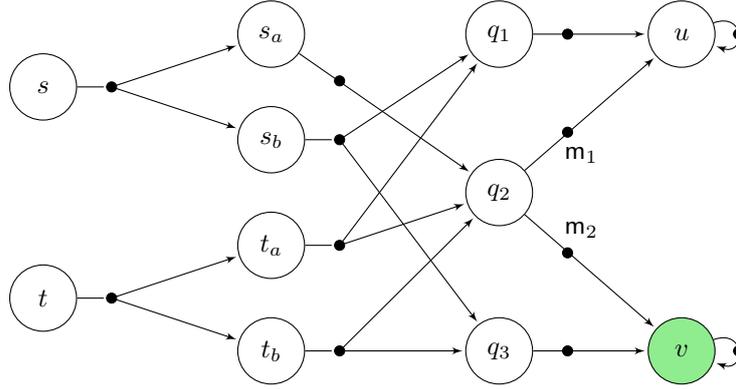
If the instance is positive, there exists a vector  $\mathbf{b} \in \mathcal{B}(P)$  such that  $\mu \mathbf{b} \neq \nu \mathbf{b}$ . Since  $\mathbf{b}$  is an MD vector which corresponds to a pair  $(\alpha, w) \in P$ , we have  $\mu M_\alpha(w) \mathbf{1} \neq \nu M_\alpha(w) \mathbf{1}$ , equivalently  $\Pr_{\mu, \mathcal{D}(\alpha)}(\text{Run}(w)) \neq \Pr_{\nu, \mathcal{D}(\alpha)}(\text{Run}(w))$ .  $\blacktriangleleft$

## 4 Probabilistic Bisimulation Inequivalence

In this section we show that one can decide in polynomial time whether there exists a memoryless strategy  $\alpha$  so that  $s \not\sim t$  in  $\mathcal{D}(\alpha)$ , i.e., we show that  $\text{PB} > 0$  is in P.

For some MDPs, there might be memoryless strategies such that  $s \not\sim t$  in the induced LMC but no such strategy is MD. The MDP in Figure 1 is such an example. Similar to the *or-gate* construction of [8, Theorem 2], we have  $s \sim t$  if and only if  $q_1 \sim q_2$  or  $q_2 \sim q_3$ . We have  $q_2 \sim q_1$  if the MD strategy maps  $q_2$  to the action that goes to state  $u$ , otherwise  $q_2 \sim q_3$  if the MD strategy maps  $q_2$  to the action that goes to state  $v$ . This rules out the algorithm that goes through all the MD strategies.

We define an equivalence relation and run the classical polynomial-time partition refinement as shown in Algorithm 2, with an equivalence relation  $\equiv_X$  defined below. At the beginning, all states are in the same equivalence class. In a refinement step, a pair of states is split if there *could* exist a memoryless strategy that makes them not probabilistic bisimilar. Two states  $s, t$  remain in the same equivalence class until the end if and only if they are probabilistic bisimilar under all memoryless strategies.



■ **Figure 1** In this MDP no MD strategy witnesses  $s \not\sim t$ . All states have the same label except state  $v$ . By default the transition probabilities out of each action are uniformly distributed.

■ **Algorithm 2** Partition Refinement.

---

```

1  $i = 0; X_0 := \{S\}$ 
2 repeat
3    $i := i + 1$ 
4    $X_i := S / \equiv_{X_{i-1}}$ 
5 until  $X_i = X_{i-1}$ 
    
```

---

The correctness of this approach is not obvious, as some splits that occurred in different iterations of the algorithm may have been due to different, potentially contradictory, memoryless strategies. Furthermore, the algorithm does not compute a memoryless strategy that witnesses  $s \not\sim t$ . The key to solving both problems will be Lemma 11.

A partition of the states  $S$  is a set  $X$  consisting of pairwise disjoint subsets  $E$  of  $S$  with  $\bigcup_{E \in X} E = S$ . Recall that  $\varphi(s, m)(s')$  is the transition probability from  $s$  to  $s'$  when choosing action  $m$ . Similarly,  $\varphi(s, m)(E)$  is the transition probability from  $s$  to  $E \subseteq S$  when choosing action  $m$ . We write  $\varphi(s, m)(X)$  to denote the vector (probability distribution)  $(\varphi(s, m)(E))_{E \in X}$ . We define  $\varphi(s)(X) = \{\varphi(s, m)(X) : m \in \mathcal{A}(s)\}$ , which is a set of probabilistic distributions over the partition  $X$  when choosing all available actions of  $s$ . Each partition is associated with an equivalence relation  $\equiv_X$  on  $S$ :  $s \equiv_X s'$  if and only if

- $\ell(s) = \ell(s')$ ;
- $s \neq s' \implies |\varphi(s)(X)| = 1$  and  $\varphi(s)(X) = \varphi(s')(X)$ .

Let  $S / \equiv_X$  denote the set of equivalence classes with respect to  $\equiv_X$ , which forms a partition of  $S$ . We present in Table 2 the partitions of running the algorithm on the MDP in Figure 1. Notice that states  $s$  and  $t$  are no longer in the same equivalence class at the end.

■ **Table 2** Example of running Algorithm 2 on the MDP in Figure 1.

---

$X_0 = \{S\}$
$X_1 = \{\{v\}, S \setminus \{v\}\}$
$X_2 = \{\{v\}, \{q_2\}, \{q_3\}, S \setminus \{v, q_2, q_3\}\}$
$X_3 = \{\{v\}, \{q_2\}, \{q_3\}, \{s_a\}, \{s_b\}, \{t_a\}, \{t_b\}, \{s, t, q_1, u\}\}$
$X_4 = \{\{v\}, \{q_2\}, \{q_3\}, \{s_a\}, \{s_b\}, \{t_a\}, \{t_b\}, \{s\}, \{t\}, \{q_1, u\}\}$

---

The following lemma is standard, and claims that the partition gets finer.

► **Lemma 7.** *For all  $i \in \mathbb{N}$ , we have  $\equiv_{X_{i+1}} \subseteq \equiv_{X_i}$ .*

If the loop in Algorithm 2 is performed  $|S| - 1$  times then  $X_{|S|-1}$  consists of  $|S|$  one-element sets. Hence at most after  $|S| - 1$  refinement steps the partition  $X_i$  cannot be refined. We aim at proving that  $s \equiv_{X_{|S|-1}} t$  if and only if  $s \sim_{\mathcal{D}(\alpha)} t$  for all memoryless strategies  $\alpha$ . In the following lemma we show the forward direction:

► **Lemma 8.** *Let  $X$  be a partition and  $X = S/\equiv_X$ . We have  $\equiv_X \subseteq \sim_{\mathcal{D}(\alpha)}$  for all memoryless strategies  $\alpha$ .*

For the converse, to guarantee  $\equiv_{X_{|S|-1}}$  is not too fine, it suffices to show that there exists a memoryless strategy  $\alpha'$  such that  $\sim_{\mathcal{D}(\alpha')} \subseteq \equiv_X$  where  $X = S/\equiv_X$ . To do that, we define the equivalence relations  $\sim_{\mathcal{D}(\alpha)}^i$  with  $0 \leq i \leq |S|$  for all memoryless strategies  $\alpha$ .

Let  $\alpha$  be a memoryless strategy. Let  $\tau$  be the transition function for the LMC  $\mathcal{D}(\alpha)$ . Define the equivalence relation  $\sim_{\mathcal{D}(\alpha)}^i$  with  $0 \leq i \leq |S|$  on  $S$ :  $s \sim_{\mathcal{D}(\alpha)}^i s'$  if and only if

- $\ell(s) = \ell(s')$ ;
- $i > 0 \implies \tau(s)(E) = \tau(s')(E)$  for all  $E \in S/\sim_{\mathcal{D}(\alpha)}^{i-1}$ .

Note that for the LMC  $\mathcal{D}(\alpha)$ , we have  $\sim_{\mathcal{D}(\alpha)}^{i+1} \subseteq \sim_{\mathcal{D}(\alpha)}^i$  for all  $i \in \mathbb{N}$  and  $\sim_{\mathcal{D}(\alpha)}^{|S|-1}$  is the probabilistic bisimilarity for the LMC  $\mathcal{D}(\alpha)$  (see, e.g., [2]).

Since the witness strategy might not be MD, we compute a set of prime numbers that can be used to form the weights of the actions. The prime numbers are used to rule out certain “accidental” bisimulations. We denote by  $\text{size}(\mathcal{D})$  the size of the representation of an object  $\mathcal{D}$ . We represent rational numbers as quotients of integers written in binary.

For  $u \in S$ ,  $m \in \mathcal{A}(u)$  and  $E \subseteq S$ , we express  $\varphi(u, m)(E)$  as an irreducible fraction  $\frac{a_{u,m,E}}{b_{u,m,E}}$  where  $a_{u,m,E}$  and  $b_{u,m,E}$  are coprime integers. Similarly, for  $u \in S$ ,  $m_1, m_2 \in \mathcal{A}(u)$  and  $E \subseteq S$ ,  $\varphi(u, m_1)(E) - \varphi(u, m_2)(E)$  is expressed as an irreducible fraction  $\frac{c_{u,m_1,m_2,E}}{d_{u,m_1,m_2,E}}$  that  $c_{u,m_1,m_2,E}$  and  $d_{u,m_1,m_2,E}$  are coprime integers. Let  $N \subseteq \mathbb{N}$  be the following set:

$$N = \{b_{u,m,E} : u \in S, m \in \mathcal{A}(u) \text{ and } E \in \bigcup_i X_i\} \cup \\ \{c_{u,m_1,m_2,E} : u \in S, m_1, m_2 \in \mathcal{A}(u), E \in \bigcup_i X_i \text{ and } c_{u,m_1,m_2,E} > 0\}.$$

We denote by  $\theta(x)$  the number of different prime factors of a positive integer  $x$ , and by  $\theta(N)$  the number of different prime factors in  $N$  where  $N$  is a set of positive integers.

► **Lemma 9.**  *$\theta(N)$  is polynomial in  $\text{size}(\mathcal{D})$ .*

Using the prime number theorem, we obtain the following lemma which guarantees that one can find  $|S|$  extra different prime numbers other than the prime factors in  $N$  in time polynomial in  $\text{size}(\mathcal{D})$ .

► **Lemma 10.** *One can find  $|S|$  different prime numbers in time polynomial in  $\text{size}(\mathcal{D})$  such that any of them is coprime to all numbers in the set  $N$ .*

To each  $u \in S$ , we assign a different prime number  $p_u$  that is coprime with all  $b \in N$ . This can be done in polynomial time by Lemma 10. We have

$$p_u \nmid b \text{ for all } b \in N \quad \text{and} \quad u \neq v \implies p_u \neq p_v \text{ for all } u, v \in S \quad (2)$$

We define a partial memoryless strategy for  $\mathcal{D}$  to be a partial function  $\alpha' : S \rightarrow \text{Distr}(\mathcal{A})$  that, given a state  $s \in S$ , returns  $\alpha'(s) \in \text{Distr}(\mathcal{A}(s))$  if  $\alpha'(s)$  is defined. A memoryless strategy  $\alpha$  is compatible with a partial memoryless strategy  $\alpha'$ , written as  $\alpha \sqsupseteq \alpha'$ , if and only if  $\alpha(s) = \alpha'(s)$  for all  $s$  such that  $\alpha'(s)$  is defined. We construct the partial memoryless strategy iteratively.

## 49:10 Comparing Labelled Markov Decision Processes

► **Lemma 11.** *Let  $i \in \mathbb{N}$  with  $i \leq |S|$ . One can compute in polynomial time a partial strategy  $\alpha'_i$  such that  $\sim_{\mathcal{D}(\alpha)}^i \subseteq \equiv_{X_i}$  for all  $\alpha \sqsupseteq \alpha'_i$ .*

**Proof sketch.** We prove the statement by induction on  $i$ . Let  $s, t \in S$ . The base case is  $i = 0$ . By definition, we have if  $s \not\equiv_{X_0} t$  then  $\ell(s) \neq \ell(t)$ . We also have if  $\ell(s) \neq \ell(t)$ , then  $s \not\sim_{\mathcal{D}(\alpha)}^0 t$  in  $\mathcal{D}(\alpha)$  for all memoryless strategy  $\alpha$ . We simply let  $\alpha'_0$  be the empty partial function such that  $\alpha \sqsupseteq \alpha'_0$  holds for any memoryless strategy  $\alpha$ .

For the induction step, assume that we can compute in polynomial time a partial strategy  $\alpha'_i$  such that  $\sim_{\mathcal{D}(\alpha)}^i \subseteq \equiv_{X_i}$  for all  $\alpha \sqsupseteq \alpha'_i$ , i.e., if  $s \not\equiv_{X_i} t$  then  $s \not\sim_{\mathcal{D}(\alpha)}^i t$  in  $\mathcal{D}(\alpha)$ . We show the statement holds for  $i + 1$ .

■ **Algorithm 3** Polynomial-time algorithm constructing  $\alpha'_{i+1}$ .

---

```

1  $\alpha'_{i+1} := \alpha'_i$ 
2 foreach  $u \in S$  such that  $|\varphi(u)(X_i)| = 1$  and  $|\varphi(u)(X_{i+1})| \neq 1$  do
3   | pick  $\mathbf{m}_1, \mathbf{m}_2 \in \mathcal{A}(u)$  such that for a set  $E \in X_{i+1} : \varphi(u, \mathbf{m}_1)(E) > \varphi(u, \mathbf{m}_2)(E)$ 
4   |  $\alpha'_{i+1}(u)(\mathbf{m}_1) := \frac{1}{p_u}$ 
5   |  $\alpha'_{i+1}(u)(\mathbf{m}_2) := 1 - \frac{1}{p_u}$ 
6 end

```

---

Algorithm 3 computes the partial memoryless strategy  $\alpha'_{i+1}$  in polynomial time. We show that  $\alpha'_j$  does not overwrite  $\alpha'_k$  for all  $k < j$ . It follows that for any  $\alpha \sqsupseteq \alpha'_{i+1}$ , it satisfies  $\alpha \sqsupseteq \alpha'_i$ . Let  $\alpha \sqsupseteq \alpha'_{i+1}$ . Assume  $s \not\equiv_{X_{i+1}} t$ . We distinguish the two cases:  $s \not\sim_{\mathcal{D}(\alpha)}^i t$  and  $s \sim_{\mathcal{D}(\alpha)}^i t$ . For both cases we can derive  $s \not\sim_{\mathcal{D}(\alpha)}^{i+1} t$ , i.e.,  $\sim_{\mathcal{D}(\alpha)}^{i+1} \subseteq \equiv_{X_{i+1}}$  as desired. The details can be found in [24]. ◀

For example, let  $p_{q_2}$ , the prime number assigned to state  $q_2$  in Figure 1, be 3 which is coprime with numbers in  $N = \{1, 2\}$ .<sup>1</sup> We show how the partial strategy  $\alpha'_1$  is constructed. On line 1 of Algorithm 3,  $\alpha'_1$  is equal to  $\alpha'_0$ , the empty partial function. Since  $|\varphi(q_2)(X_0)| = 1$  and  $|\varphi(q_2)(X_1)| = 2$ , we enter the for loop. We can pick  $\mathbf{m}_1, \mathbf{m}_2 \in \mathcal{A}(q_2)$  and  $E = S \setminus \{v\} \in X_1$  on line 3, since  $\varphi(q_2, \mathbf{m}_1)(E) = 1 > 0 = \varphi(q_2, \mathbf{m}_2)(E)$ . We then define the strategy for  $q_2$  (line 4 and 5):  $\alpha'_1(q_2)(\mathbf{m}_1) = \frac{1}{3}$  and  $\alpha'_1(q_2)(\mathbf{m}_2) = \frac{2}{3}$ . We have completed the construction of  $\alpha'_1$  as  $|\varphi(u)(X_0)| = |\varphi(u)(X_1)| = 1$  for all other state  $u$ .

► **Theorem 12.** *One can compute in polynomial time a memoryless strategy  $\beta$  such that  $\sim_{\mathcal{D}(\beta)} \subseteq \sim_{\mathcal{D}(\alpha)}$  for all memoryless strategies  $\alpha$ .*

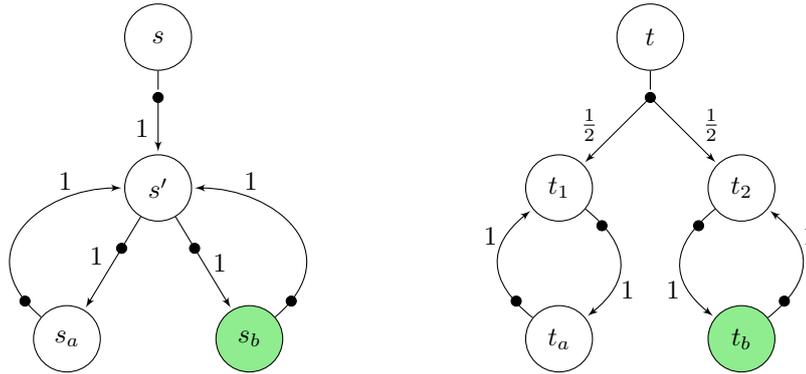
**Proof.** By invoking Lemma 11 for  $i = |S| - 1$ , a partial strategy  $\alpha'_{|S|-1}$  can be computed in polynomial time such that  $\sim_{\mathcal{D}(\alpha)}^{|S|-1} \subseteq \equiv_{X_{|S|-1}}$  for all  $\alpha \sqsupseteq \alpha'_{|S|-1}$ . Since  $\sim_{\mathcal{D}(\alpha)}^{|S|-1} = \sim_{\mathcal{D}(\alpha)}$ , we have  $\sim_{\mathcal{D}(\alpha)} \subseteq \equiv_{X_{|S|-1}}$  for all  $\alpha \sqsupseteq \alpha'_{|S|-1}$ . Let  $\beta$  be a memoryless strategy defined by

$$\beta(u) = \begin{cases} \alpha'_{|S|-1}(u) & \text{if } \alpha'_{|S|-1}(u) \text{ is defined} \\ \delta_{\mathbf{m}_u} \text{ where } \mathbf{m}_u \in \mathcal{A}(u) & \text{otherwise} \end{cases}$$

By definition the memoryless strategy  $\beta$  is compatible with  $\alpha'_{|S|-1}$ . We have:

$$\begin{aligned} \sim_{\mathcal{D}(\beta)} &\subseteq \equiv_{X_{|S|-1}} & \beta &\sqsupseteq \alpha'_{|S|-1} \\ &\subseteq \sim_{\mathcal{D}(\alpha)} \text{ for all strategy } \alpha & X_{|S|-1} &= S / \equiv_{X_{|S|-1}} \text{ and Lemma 8} \end{aligned} \quad \blacktriangleleft$$

<sup>1</sup> We have  $2 \in N$  since  $\varphi(s, \mathbf{m}_s)(\{s_a\}) = \frac{1}{2}$  where  $\mathbf{m}_s$  is the only available action at state  $s$ .



■ **Figure 2** In this MDP, no MD strategy witnesses  $d_{tv}(\delta_s, \delta_t) = 1$  (nor  $d_{pb}(s, t) = 1$ ). States  $s_b$  and  $t_b$  have label  $b$  while all other states have label  $a$ .

► **Corollary 13.** *The problem  $PB > 0$  is in P. Further, for any positive instance of the problem  $PB > 0$ , we can compute in polynomial time a memoryless strategy that witnesses  $s \not\sim t$ .*

## 5 The Distance One Problems

In this section, we summarise the results for the two distance one problems, namely  $TV = 1$  and  $PB = 1$ . The *existential theory of the reals*,  $ETR$ , is the set of valid formulas of the form

$$\exists x_1 \dots \exists x_n R(x_1, \dots, x_n),$$

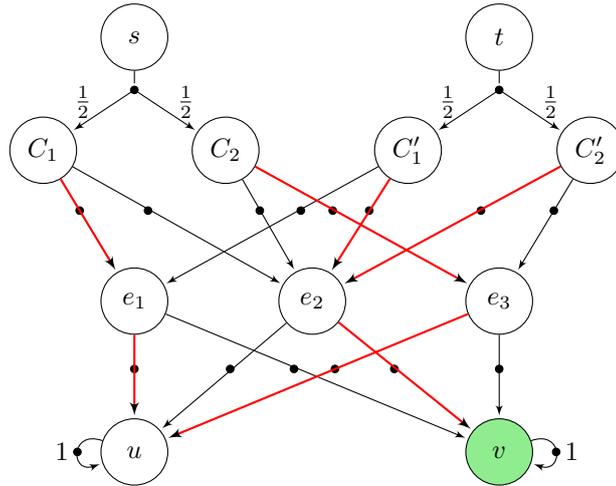
where  $R$  is a boolean combination of comparisons of the form  $p(x_1, \dots, x_n) \sim 0$ , in which  $p(x_1, \dots, x_n)$  is a multivariate polynomial (with rational coefficients) and  $\sim \in \{<, >, \leq, \geq, =, \neq\}$ . The complexity class  $\exists\mathbb{R}$  [32] consists of those problems that are many-one reducible to  $ETR$  in polynomial time. Since  $ETR$  is NP-hard and in PSPACE [5, 31], we have  $NP \subseteq \exists\mathbb{R} \subseteq PSPACE$ .

For some MDPs there exist memoryless strategies that make  $d_{tv}(\delta_s, \delta_t) = 1$  but no such strategy is MD. For example, consider the MDP in Figure 2 which has two MD strategies. We have  $d_{tv}(\delta_s, \delta_t) = \frac{1}{2}$  which is less than 1 in the LMC induced by any of the two MD strategies, and  $d_{tv}(\delta_s, \delta_t) = 1$  in the LMC induced by any other strategy. Thus, we cannot simply guess an MD strategy. We show that the problem  $TV = 1$  is in  $\exists\mathbb{R}$ , using the characterization from [9, Theorem 21] of total variation distance 1 in LMCs and some reasoning on convex polyhedra:

► **Theorem 14.** *The problem  $TV = 1$  is in  $\exists\mathbb{R}$ .*

The problem  $TV = 1$  is NP-hard, and  $PB = 1$  is NP-complete. The hardness results for both problems are by reductions from the Set Splitting problem. Given a finite set  $S$  and a collection  $\mathcal{C}$  of subsets of  $S$ , Set Splitting asks whether there is a partition of  $S$  into disjoint sets  $S_1$  and  $S_2$  such that no set in  $\mathcal{C}$  is a subset of  $S_1$  or  $S_2$ .

Let  $\langle S, \mathcal{C} \rangle$  be an instance of Set Splitting where  $S = \{e_1, \dots, e_n\}$  and  $\mathcal{C} = \{C_1, \dots, C_m\}$  is a collection of subsets of  $S$ . We construct an MDP  $\mathcal{D}$  consisting of the following states: two states  $s$  and  $t$ , a state  $e_i$  for each element in  $S$ , twin states  $C_j$  and  $C'_j$  for each element in  $\mathcal{C}$ , two sink states  $u$  and  $v$ . State  $v$  has label  $b$  while all other states have label  $a$ . State  $s$  ( $t$ ) has a single action which goes with uniform probability  $\frac{1}{m}$  to states  $C_i$  ( $C'_i$ ) for  $1 \leq i \leq m$ . For each



■ **Figure 3** The MDP in the reduction from Set Splitting for NP-hardness of  $TV = 1$  (or  $PB = 1$ ).

$e_i \in C_j$ , there is an action from state  $C_j$  and  $C'_j$  leading to state  $e_i$  with probability one. Each state  $e_i$  has two actions going to the sink states  $u$  and  $v$  with probability one, respectively. We have:  $\langle S, \mathcal{C} \rangle \in \text{Set Splitting} \iff \exists$  memoryless strategy  $\alpha$  such that  $d_{tv}(\delta_s, \delta_t) = 1$  in  $\mathcal{D}(\alpha)$ .

For example, let  $S = \{e_1, e_2, e_3\}$  and  $\mathcal{C} = \{C_1, C_2\}$  with  $C_1 = \{e_1, e_2\}$  and  $C_2 = \{e_2, e_3\}$ . Figure 3 shows the corresponding MDP. The MD strategy highlighted, corresponding to the partition of  $S_1 = \{e_1, e_3\}$  and  $S_2 = \{e_2\}$ , witnesses  $d_{tv}(\delta_s, \delta_t) = 1$ .

► **Theorem 15.** *The Set Splitting problem is polynomial-time many-one reducible to  $TV = 1$ , hence  $TV = 1$  is NP-hard.*

The problem  $PB = 1$  is NP-complete. The MDP in Figure 2 is also an example of no MD strategy witnessing  $d_{pb}(s, t) = 1$ , which rules out the algorithm of simply guessing an MD strategy. By [36], deciding whether  $d_{pb}(s, t) = 1$  in an LMC can be formulated as a reachability problem on a directed graph induced by the LMC. One can nondeterministically guess the graph induced by the LMC and use Algorithm 3 to construct a memoryless strategy that witnesses  $d_{pb}(s, t) = 1$ .

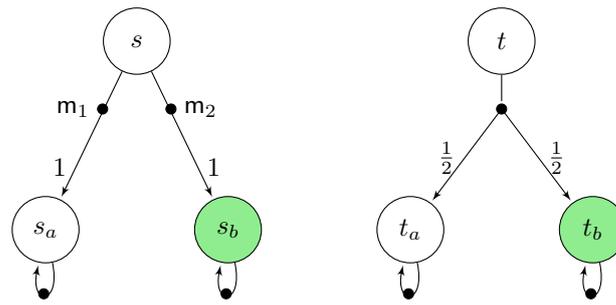
► **Theorem 16.** *The problem  $PB = 1$  is NP-complete.*

## 6 Making Distances Small

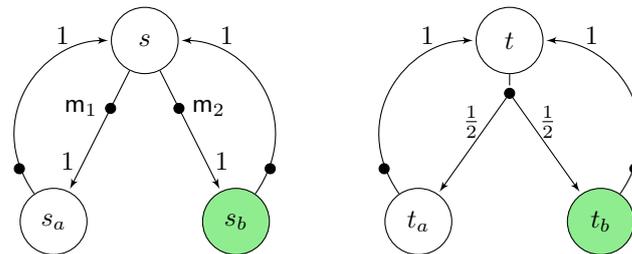
In this section, we summarise the results for the remaining problems, which are all about making the distance small (equal to 0 or less than 1).

We show that  $TV = 0$  and  $TV < 1$  are  $\exists\mathbb{R}$ -complete. The proof for the membership of  $TV = 0$  in  $\exists\mathbb{R}$  is similar to [17, Theorem 4.3]. For both hardness results we provide reductions from the *Nonnegative Matrix Factorization (NMF)* problem, which asks, given a nonnegative matrix  $J \in \mathbb{Q}^{n \times m}$  and a number  $r \in \mathbb{N}$ , whether there exists a factorization  $J = A \cdot W$  with nonnegative matrices  $A \in \mathbb{R}^{n \times r}$  and  $W \in \mathbb{R}^{r \times m}$ . The NMF problem is  $\exists\mathbb{R}$ -complete by [35, Theorem 2], see also [10, 40, 1] for more details on the NMF problem. The reduction is similar to [17, Theorem 4.5].

► **Theorem 17.** *The problem  $TV = 0$  is  $\exists\mathbb{R}$ -complete.*



■ **Figure 4** In this MDP, no MD strategy witnesses  $d_{pb}(s, t) = 0$ . States  $s_b$  and  $t_b$  have label  $b$  while all other states have label  $a$ .



■ **Figure 5** In this MDP, no MD strategy witnesses  $d_{pb}(s, t) < 1$ . States  $s_b$  and  $t_b$  have label  $b$  while all other states have label  $a$ .

► **Theorem 18.** *The problem  $TV < 1$  is  $\exists\mathbb{R}$ -complete.*

Finally, we show that  $PB = 0$  and  $PB < 1$  are NP-complete. For some MDPs there exist memoryless strategies that make  $d_{pb}(s, t) = 0$  (resp.  $d_{pb}(s, t) < 1$ ) but no such strategy is MD. Indeed, for the MDP in Figure 4 (resp. Figure 5), it is easy to check that the only strategy  $\alpha$  which makes  $d_{pb}(s, t) = 0$  (resp.  $d_{pb}(s, t) < 1$ ), requires randomness, that is,  $\alpha(s)(m_1) = \alpha(s)(m_2) = \frac{1}{2}$ , where  $m_1$  and  $m_2$  are the two available actions of state  $s$ . Thus, to show the NP upper bound, we cannot simply guess an MD strategy. Instead, one can nondeterministically guess a partition of the states and check in polynomial time if the partition is a probabilistic bisimulation.

The hardness results for both problems are by reductions from the Subset Sum problem. The reduction is similar to [17, Theorem 4.1].

► **Theorem 19.** *The problem  $PB = 0$  is NP-complete.*

By [36], deciding whether  $d_{pb}(s, t) < 1$  in an LMC can be formulated as a reachability problem on a directed graph induced by the LMC. In addition to a partition, our NP algorithm also guesses the graph induced by the LMC.

► **Theorem 20.** *The problem  $PB < 1$  is NP-complete.*

## 7 Conclusions

We have studied the computational complexity of qualitative comparison problems in labelled MDPs. Motivated by the connection between obliviousness/anonymity and equivalence, we have devised polynomial-time algorithms to decide the existence of strategies for trace and bisimulation *inequivalence*. In case of trace inequivalence, there always exists an MD witness

strategy, and our algorithm computes it. The trace inequivalence algorithm is based on linear-algebra arguments that are considerably more subtle than in the LMC case. For bisimulation inequivalence, MD strategies may not exist, but we have devised a polynomial-time algorithm to compute a memoryless strategy witnessing inequivalence; here the randomization is based on prime numbers to rule out certain “accidental” bisimulations. The other 6 problems do not have polynomial complexity (unless  $P = NP$ ), and we have established completeness results for all of them except  $TV = 1$ , where a complexity gap between  $NP$  and  $\exists\mathbb{R}$  remains.

Concerning the relationship to interval Markov chains and parametric Markov chains mentioned in the introduction, the lower complexity bounds that we have derived in this paper carry over to corresponding problems in these models. Transferring the upper bounds requires additional work, as, e.g., even the consistency problem for IMCs (i.e., whether there *exists* a Markov chain conforming to an IMC) is not obvious to solve. Nevertheless, the algorithmic insights of this paper will be needed.

---

### References

- 1 Sanjeev Arora, Rong Ge, Ravi Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization - provably. In *STOC*, pages 145–162. ACM, 2012.
- 2 Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 50–61, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- 3 Michael Benedikt, Rastislav Lenhardt, and James Worrell. LTL model checking of interval Markov chains. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2013.
- 4 Patrick Billingsley. *Probability and measure*. Wiley Series in Probability and Statistics. Wiley, New York, NY, USA, 3rd edition, 1995.
- 5 John Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pages 460–467, 1988.
- 6 Souymodip Chakraborty and Joost-Pieter Katoen. Model checking of open interval Markov chains. In Marco Gribaudo, Daniele Manini, and Anne Remke, editors, *Analytical and Stochastic Modelling Techniques and Applications*, pages 30–42. Springer International Publishing, 2015.
- 7 Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.
- 8 Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In Lars Birkedal, editor, *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451, Tallinn, Estonia, March/April 2012. Springer-Verlag.
- 9 Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, New York, NY, USA, 2014*. ACM.
- 10 Joel E. Cohen and Uriel G. Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, 190:149–168, 1993.
- 11 Benoît Delahaye. Consistency for parametric interval markov chains. In Étienne André and Goran Frehse, editors, *2nd International Workshop on Synthesis of Complex Parameters*,

- SymCoP 2015, April 11, 2015, London, United Kingdom*, volume 44 of *OASICS*, pages 17–32. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
- 12 Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Decision problems for interval Markov chains. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*, volume 6638 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 2011.
  - 13 Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.
  - 14 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In Jos Baeten and Sjouke Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 258–273, Eindhoven, The Netherlands, August 1999. Springer-Verlag.
  - 15 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004.
  - 16 L. Doyen, T.A. Henzinger, and J.-F. Raskin. Equivalence of labeled Markov chains. *International Journal on Foundations of Computer Science*, 19(3):549–563, 2008.
  - 17 Nathanaël Fijalkow, Stefan Kiefer, and Mahsa Shirmohammadi. Trace refinement in labelled Markov decision processes. *Logical Methods in Computer Science*, 16(2), 2020.
  - 18 Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Int. J. Softw. Tools Technol. Transf.*, 13(1):3–19, 2011.
  - 19 Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker Storm, 2020. [arXiv:arXiv:2002.07080](https://arxiv.org/abs/2002.07080).
  - 20 Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 266–277. IEEE Computer Society, 1991.
  - 21 John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer, 1976.
  - 22 S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *CAV*, volume 6806 of *LNCS*, pages 526–540. Springer, 2011.
  - 23 S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. APEX: An analyzer for open probabilistic programs. In *CAV*, volume 7358 of *LNCS*, pages 693–698. Springer, 2012.
  - 24 Stefan Kiefer and Qiyi Tang. Comparing labelled markov decision processes, 2020. [arXiv:2009.11643](https://arxiv.org/abs/2009.11643).
  - 25 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
  - 26 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
  - 27 L. Li and Y. Feng. Quantum Markov chains: Description of hybrid systems, decidability of equivalence, and model checking linear-time properties. *Information and Computation*, 244:229–244, 2015.
  - 28 T.M. Ngo, M. Stoelinga, and M. Huisman. Confidentiality for probabilistic multi-threaded programs and its verification. In *Engineering Secure Software and Systems*, volume 7781 of *LNCS*, pages 107–122. Springer, 2013.
  - 29 A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
  - 30 S. Peyronnet, M. de Rougemont, and Y. Strobecki. Approximate verification and enumeration problems. In *ICTAC*, volume 7521 of *LNCS*, pages 228–242. Springer, 2012.
  - 31 James Renegar. On the computational complexity and geometry of the first-order theory of the reals. Parts I–III. *Journal of Symbolic Computation*, 13(3):255–352, 1992.

- 32 Marcus Schaefer and Daniel Stefankovic. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory Comput. Syst.*, 60(2):172–193, 2017. doi:10.1007/s00224-015-9662-0.
- 33 M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
- 34 Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking markov chains in the presence of uncertainties. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006.
- 35 Yaroslav Shitov. A universality theorem for nonnegative matrix factorizations, 2016. arXiv:1606.09068.
- 36 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for labelled Markov chains. In Hana Chockler and Georg Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification*, volume 10981 of *Lecture Notes in Computer Science*, pages 681–699, Oxford, UK, July 2018. Springer-Verlag. doi:10.1007/978-3-319-96145-3\_39.
- 37 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for probabilistic automata. *Journal of Computer and System Sciences*, 111:57–84, 2020.
- 38 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.
- 39 Antti Valmari and Giuliana Franceschinis. Simple  $O(m \log n)$  time Markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2010.
- 40 Stephen A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009.
- 41 Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen. On the complexity of reachability in parametric markov decision processes. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.14.

# Computable Analysis for Verified Exact Real Computation

Michal Konečný 

School of Engineering and Applied Science, Aston University, UK

m.konecny@aston.ac.uk

Florian Steinberg

Inria Saclay, France

fsteinberg@gmail.com

Holger Thies 

Department of Informatics, Kyushu University, Japan

thies@inf.kyushu-u.ac.jp

---

## Abstract

---

We use ideas from computable analysis to formalize exact real number computation in the COQ proof assistant. Our formalization is built on top of the INCONE library, a COQ library for computable analysis. We use the theoretical framework that computable analysis provides to systematically generate target specifications for real number algorithms. First we give very simple algorithms that fulfill these specifications based on rational approximations. To provide more efficient algorithms, we develop alternate representations that utilize an existing formalization of floating-point algorithms and interval arithmetic in combination with methods used by software packages for exact real arithmetic that focus on execution speed. We also define a general framework to define real number algorithms independently of their concrete encoding and to prove them correct. Algorithms verified in our framework can be extracted to Haskell programs for efficient computation. The performance of the extracted code is comparable to programs produced using non-verified software packages. This is without the need to optimize the extracted code by hand.

As an example, we formalize an algorithm for the square root function based on the Heron method. The algorithm is parametric in the implementation of the real number datatype, not referring to any details of its implementation. Thus the same verified algorithm can be used with different real number representations. Since Boolean valued comparisons of real numbers are not decidable, our algorithms use basic operations that take values in the Kleeneans and Sierpinski space. We develop some of the theory of these spaces. To capture the semantics of non-sequential operations, such as the “parallel or”, we use multivalued functions.

**2012 ACM Subject Classification** Theory of computation → Logic and verification; Mathematics of computing → Continuous mathematics

**Keywords and phrases** Computable Analysis, exact real computation, formal proofs, proof assistant, Coq

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.50

**Supplementary Material** The incone library at <https://github.com/FlorianSteinberg/incone>

**Funding** *Michal Konečný*: This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

*Holger Thies*: Supported by JSPS KAKENHI Grant Numbers JP18J10407 and JP20K19744 and by the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks).



© Michal Konečný, Florian Steinberg, and Holger Thies;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 50; pp. 50:1–50:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Computable analysis is a formal model for computation on real numbers and other spaces of interest in analysis [25, 9]. It extends classical computability theory from discrete structures to continuous ones. The model of computation used in computable analysis operates on properly infinite data while being realistic in the sense that proofs of computability specify algorithms that can in principle be implemented. Software for computation on the reals based on ideas from computable analysis is often labeled as *exact real computation* as such software allows to approximate real number outputs up to any desired precision. In practice, this can be realized in different ways and several implementations exist [22, 17, 3, 13]. In contrast to implementations using floating-point arithmetic, algorithms from computable analysis have sound compositional semantics and come with a mathematical correctness proof, making them well-suited for safety-critical applications. Proof assistants and formal methods are increasingly used to verify the correctness of such software and computable analysis goes well with this kind of verification.

In this work we present a new and fully formally verified implementation of exact real computation in COQ that makes use of COQ's code extraction features to generate efficient Haskell code for algorithms written and verified inside the proof assistant. The work builds on the INCOME library, a formalization of ideas from computable analysis in COQ [24]. Implementations of exact real computation usually hide the internal details of the encoding from the user and instead provide a set of basic operations on real numbers that can be used to build more complicated algorithms. We follow this approach by defining a structure for basic operations on real numbers. Instantiating this structure means to explicitly give an encoding of the reals and algorithms for the basic operations and proving them correct. More complicated operations can then be defined using tools for composing functions that are available in the INCOME library. Correctness proofs can be made independent of the concrete representation and different representations can be exchanged and compared easily. As algorithms verified in the proof assistant can be extracted to efficient Haskell code we hope that our work allows developers of exact real computation libraries to verify some particularly critical fragments and easily integrate the generated code into the library.

Computing with real numbers is central in many applications. It should therefore not be surprising that a treatment of the reals is available in most modern proof assistants [6]. In the COQ proof assistant in particular there exist a wide range of work covering the spectrum from purely mathematical and inherently non-computational [5, 2] to verification of approximate computations and concrete error bounds [7, 20].

In this work we treat the real numbers as a represented space. A represented space is an infinite data type that is both exact and fully computational but reasoned about using classical mathematics. Our work is by far not the first implementation of fully computational reals in a proof assistant, or even in COQ. A popular implementation is the C-CoRN library [12] which is based on constructive mathematics. Working constructively has the advantage that every proof has computational content. A constructive proof of an existential statement, for instance, gives rise to an algorithm to compute said object. The price to pay is that a constructive proof is harder to find and this extra effort may not be worthwhile in particular for proofs of correctness, where the computational content is of little to no interest. Most mathematicians and computer scientists distinguish formulation of algorithms from proving its correctness, and prefer the use of classical reasoning for the latter.

Our work and the INCOME library are based on computable analysis which is a part of classical mathematics. For our implementation this means that we use the classical axiomatization of the reals from COQ's standard library for specification. Computational

content is added in a second step through the use of encodings over certain spaces of functions and the formulation of algorithms on these. Thus, there is a clear separation between the formulation of an algorithm that operates only on computationally meaningful objects and its (possibly non-constructive) correctness proof that may involve purely mathematical objects such as abstract real numbers. We consider this a more pragmatic approach towards computational reasoning over mathematical structures and hope that it can be appealing to classically trained mathematicians and computer scientists. There are also some more practical advantages of our approach. Many COQ libraries are verified against the reals from the standard library and such libraries can easily be integrated into our development. For example, we rely on a COQ library for interval arithmetic [21, 20] to be able to imitate how the most efficient non-verified packages for exact real computation operate [22, 17].

While we consider the COQ formalization one of the main contributions of this work, we keep the presentation on a more informal mathematical level and only give a short overview of the implementation in Section 6. The interested reader can find all of the source code as part of the INCONE library [23]. The parts of the library relevant for this paper are listed in Section 6 as well. A more exhaustive overview of the INCONE library can be found in [24].

## 2 Computable analysis and the Incone library

Computable analysis gives computational meaning to abstract mathematical entities such as real numbers by use of encodings over Baire space  $\mathbb{N}^{\mathbb{N}}$  called **representations** [19, 25]. To avoid an overload of coding, here and in our formal development we allow the use of arbitrary countable sets in place of the two copies of the natural numbers in Baire space. Let  $\mathbf{Q}$  and  $\mathbf{A}$  be two countable sets of **questions** and **answers** and let  $\mathcal{B} := \mathbf{A}^{\mathbf{Q}}$  be the set of functions from questions to answers. A **representation** of a set  $X$  is a partial, surjective function  $\delta: \subseteq \mathcal{B} \rightarrow X$ . For  $x \in X$ , each  $\varphi \in \mathcal{B}$  with  $\delta(\varphi) = x$  is called a **name** of  $x$  and should be understood to provide on demand information about  $x$  by assigning a valid answer to each question about  $x$ . A **represented space** is a pair  $\mathbf{X} := (X, \delta_{\mathbf{X}})$  of a set  $X$  and a representation  $\delta_{\mathbf{X}}$  of  $X$ .

A standard example is the encoding of reals by rational approximations:

► **Example 1** ( $\mathbb{R}_{\mathbb{Q}}$ : Reals via rational approximations). We denote by  $\mathbb{R}_{\mathbb{Q}}$  the represented space of the real numbers together with the representation  $\delta_{\mathbb{R}_{\mathbb{Q}}}: \subseteq \mathbb{Q}^{\mathbb{Q}} \rightarrow \mathbb{R}$  such that

$$\delta_{\mathbb{R}_{\mathbb{Q}}}(\varphi) = x \iff \forall \varepsilon > 0: |x - \varphi(\varepsilon)| \leq \varepsilon.$$

While we do not make this a formal requirement, for all of our concrete examples there exist obvious and explicitly definable bijections of  $\mathbf{Q}$  and  $\mathbf{A}$  with the natural numbers. The skeptical reader can therefore always replace the questions and answers by natural numbers to regain the classical setting from computable analysis where  $\mathcal{B}$  is only allowed to be the Baire space. Whenever we talk about computability, we assume that such bijections were fixed and refer to the well-established notion of computability of elements and of partial functions on Baire space. For instance, we call an element  $x$  of a represented space  $\mathbf{X}$  **computable** if it has a name that is computable as an element of Baire space.

Let  $\mathbf{X}$  and  $\mathbf{X}'$  be represented spaces and  $\mathcal{B} := \mathbf{A}^{\mathbf{Q}}$  denote the space of names of  $\mathbf{X}$  and  $\mathcal{B}' := \mathbf{A}'^{\mathbf{Q}'}$  that of  $\mathbf{X}'$ . We say that a function  $f: \mathbf{X} \rightarrow \mathbf{X}'$  is realized by a partial operator  $F: \subseteq \mathcal{B} \rightarrow \mathcal{B}'$  if for each name  $\varphi \in \mathcal{B}$  of some  $x \in \mathbf{X}$  the value  $F(\varphi)$  is defined and a name of  $f(x) \in \mathbf{X}'$ . As  $f$  can be called **continuous** resp. **computable** if it can be realized by such an operator, it suffices to introduce these notions for the partial operators on Baire space. For continuity we use the standard topology that Baire space comes with. Thus,

$F: \subseteq B \rightarrow \mathcal{B}'$  is continuous if for each  $q' \in \mathbf{Q}'$  its return value on a functional input  $\varphi$  is determined by a finite number of  $\varphi$ 's values. Formally, we say  $F$  is continuous if

$$\forall \varphi \in \text{dom}(F), q', \exists L \in \text{seq}(\mathbf{Q}), \forall \psi \in \text{dom}(F): \varphi|_L = \psi|_L \Rightarrow F(\varphi)(q') = F(\psi)(q')$$

where  $\text{seq}(\mathbf{Q})$  denotes the set of finite words over  $\mathbf{Q}$ . Computability is defined using oracle Turing machines [14], but we refrain from stating this definition here and assume the reader to fill this gap or use his intuition. This intuition should include that computable operations can be partial but never discontinuous.

Different representations of the same set can be compared with regards to intertranslatability, that is by asking whether the identity function is computable if the source and target spaces are equipped with the different representations. If there are continuous resp. computable translations in both directions, the spaces are isomorphic and carry the same topological resp. computability structure.

## 2.1 Specification of algorithms with multifunctions

Usually each element of a represented space has many names. Thus, it may happen that an operator returns on input of each name of an element a name of a solution of a certain problem but for different names of the same input element returns names of different solutions. In this case the algorithm solves the problem but does not realize any function on the represented spaces. This is a situation that is regularly encountered in computable analysis and a popular tool for capturing the semantics of such algorithms are multivalued functions.

A multivalued function  $\mathbf{f}: X \rightrightarrows Y$  assigns to each element  $x \in X$  a possibly empty set of eligible return values  $\mathbf{f}(x) \subseteq Y$ . Those  $x$  for which  $\mathbf{f}(x)$  is non-empty constitute the **domain**  $\text{dom}(\mathbf{f}) \subseteq X$  of  $\mathbf{f}$ . The multifunction is called **total** if its domain is all of  $X$  and **single-valued** if each value set has at most one element. A partial function can be considered a single-valued multi-function; this multifunction uniquely specifies the partial function and is total if and only if the partial function is.

A partial function  $f$  is said to **choose through** a multifunction  $\mathbf{f}$  if on each  $x \in \text{dom}(\mathbf{f})$  it returns an eligible return value, i.e.  $f(x)$  is defined and an element of  $\mathbf{f}(x)$ . Note that this allows for the domain of the partial function to be bigger than that of the multifunction. A multifunction should be considered a specification of all the partial functions that choose through it and this defines an important ordering on the multifunctions: A multifunction  $\mathbf{f}$  is said to **tighten** another multifunction  $\mathbf{g}$ , in symbols  $\mathbf{f} \prec \mathbf{g}$ , if any partial function that is a choice for  $\mathbf{f}$  is also a choice for  $\mathbf{g}$ . This can equivalently be formulated as

$$\mathbf{f} \prec \mathbf{g} \iff \text{dom}(\mathbf{g}) \subseteq \text{dom}(\mathbf{f}) \wedge \forall x \in \text{dom}(\mathbf{g}), \mathbf{f}(x) \subseteq \mathbf{g}(x).$$

If  $\mathbf{f}$  and  $\mathbf{g}$  correspond to partial functions  $f$  and  $g$  then  $\mathbf{f} \prec \mathbf{g}$  if and only if  $f$  is an extension of  $g$  and a partial function chooses through a multifunction if and only if the induced multifunction tightens it.

The multivalued functions from  $X$  to  $Y$  are in one-to-one correspondence with the relations but the natural operations on them differ from those on relations. For instance, for  $\mathbf{f}: Y \rightrightarrows Z$  and  $\mathbf{g}: X \rightrightarrows Y$  the composition as multivalued functions is defined as

$$\mathbf{f} \circ \mathbf{g}(x) := \{z \in Z \mid \mathbf{g}(x) \subseteq \text{dom}(\mathbf{f}) \wedge \exists y \in \mathbf{g}(x): z \in \mathbf{f}(y)\}.$$

This defines an associative operation that is asymmetric in contrast to the natural composition of relations which is symmetric. The multifunction composition has the advantage that it respects the interpretation as specifications. Namely, if the partial functions  $f$  and  $g$

choose through  $\mathbf{f}$  and  $\mathbf{g}$  respectively, then their composition as partial functions chooses through the composition of  $\mathbf{f}$  and  $\mathbf{g}$  as multifunctions. The multifunction composition can be characterized as returning the minimal multifunction w.r.t. tightening such that this is true and not only respects being a choice function but more generally the tightening ordering.

A multifunction  $\mathbf{f}: \mathbf{X} \rightrightarrows \mathbf{X}'$  between represented spaces  $\mathbf{X}$  and  $\mathbf{X}'$  is realized by a partial operator  $F: \subseteq \mathcal{B} \rightarrow \mathcal{B}'$  if  $F$  chooses through  $\delta_{\mathbf{X}'}^{-1} \circ \mathbf{f} \circ \delta_{\mathbf{X}}$ . Such an  $\mathbf{f}$  is called **continuous** or **computable** if it can be realized by an operator with that property. The above definition unfolds to the usual “a realizer translates each name of an element of the domain to a name of some eligible return value”.

## 2.2 The INCONE library

The INCONE library formalizes ideas from computable analysis in the COQ proof assistant closely following the outline in the previous section. The equivalent of a represented space in INCONE is called a **continuity space**. A continuity space  $\mathbf{X}$  is defined as a record consisting of an abstract type  $X$ , a space  $\mathcal{B}_{\mathbf{X}}$  of names that determines a countable inhabited type of questions  $\mathbf{Q}_{\mathbf{X}}$  and a countable type of answers  $\mathbf{A}_{\mathbf{X}}$  and finally a specification of a partial surjective function  $\delta_{\mathbf{X}}: \subseteq \mathcal{B}_{\mathbf{X}} \rightarrow X$  referred to as representation.

A number of standard constructions on represented spaces are made available by INCONE. For represented spaces  $\mathbf{X}$  and  $\mathbf{Y}$  there exists a represented space  $\mathbf{X} \times \mathbf{Y}$  whose underlying set is the Cartesian product of the sets underlying  $\mathbf{X}$  and  $\mathbf{Y}$ . Similarly, there exists a disjoint union  $\mathbf{X} + \mathbf{Y}$  of spaces and a space  $\mathbf{X}^\omega$  of infinite sequences in  $\mathbf{X}$ . There is also a space  $\mathbf{Y}^{\mathbf{X}}$  of continuous functions, but while this is interesting for possible applications it is of lesser interest for the current paper. Details about these constructions and instructions for installation and use of INCONE can be found in [24].

While INCONE defines continuity as we presented it earlier, computability is not reflected in a definition but instead captured on the meta level via COQ’s type/prop distinction. That is, an axiom-free definition of a realizer should be considered a certificate of computability of a function. While such a realizer is automatically continuous, a proof of this fact would proceed by induction on the structure of COQ terms and can clearly not be carried out internally. In principle it would be possible to extract continuity proofs using a tactic but for now the proofs have to be provided on a case by case basis by hand. Partiality is modeled using sigma types: A partial function takes as input not only an element of Baire space but also a proof that the element is contained in the domain which has to be specified beforehand. This means that the dependent type system of COQ gets involved in a meaningful way.

## 3 Finite spaces and operations on multifunctions

Besides allowing for computation on spaces of continuum cardinality, the methods of computable analysis can be used to operate on non-discrete finite spaces.

► **Example 2 (Sierpinski space)**. Consider the two-element set  $\{\top_{\mathbb{S}}, \perp_{\mathbb{S}}\}$  with the following representation: Let the questions and answers be given by  $\mathbf{Q} := \mathbb{N}$  and  $\mathbf{A} := \mathbb{B} = \{\text{true}, \text{false}\}$  and as representation use the total function  $\delta_{\mathbb{S}}$  such that

$$\delta_{\mathbb{S}}(\varphi) = \top_{\mathbb{S}} \iff \exists n, \varphi(n) = \text{true}.$$

The represented space  $\mathbb{S} := (\{\top_{\mathbb{S}}, \perp_{\mathbb{S}}\}, \delta_{\mathbb{S}})$  is called **Sierpinski space**. The elements of Sierpinski space denote convergence and divergence, respectively: For any kind of computational process with a meaningful notion of basic computational steps, we can obtain a name of an

element of Sierpinski space by saying  $\varphi(n) = \text{true}$  if the computation terminates within the first  $n$  steps and  $\varphi(n) = \text{false}$  otherwise. This reflects the interpretation of  $\top_{\mathbb{S}}$  and  $\perp_{\mathbb{S}}$ : we produce a name of  $\top_{\mathbb{S}}$  if and only if we started out with a terminating computation.

► **Example 3 (Kleeneans).** Another finite space that is important in computable analysis is the three-point set  $\{\text{true}_{\mathbb{K}}, \text{false}_{\mathbb{K}}, \perp_{\mathbb{K}}\}$  with names of type  $\mathbb{N} \rightarrow \text{opt } \mathbb{B}$  and representation

$$\delta_{\mathbb{K}}(\varphi) = \begin{cases} b_{\mathbb{K}} & \text{if } \exists n, \varphi(n) = \text{Some } b \wedge \forall m < n, \varphi(m) = \text{None} \\ \perp_{\mathbb{K}} & \text{otherwise.} \end{cases}$$

Here,  $\text{opt } \mathbf{A}$  is the disjoint union of  $\mathbf{A}$  with a single new element and for each  $a \in \mathbf{A}$  we use  $\text{Some } a$  for the corresponding element of  $\text{opt } \mathbf{A}$  and  $\text{None}$  for the new element. This space denoted by  $\mathbb{K}$  is known as the **Kleeneans** as it models Kleene's three-valued logic [4].

A continuously realizable multifunction need not have any partial continuous choice function. As example of such behavior let us consider a version of the parallel or.

► **Example 4 (The which function).** Consider the multifunction  $\text{which}: \mathbb{S} \times \mathbb{S} \rightrightarrows \mathbb{K}$  such that

$$\text{which}(s_{\text{false}}, s_{\text{true}}) := \begin{cases} \{b_{\mathbb{K}} \mid s_b = \top_{\mathbb{S}}\} & \text{if this set is non-empty} \\ \{\perp_{\mathbb{K}}\} & \text{otherwise,} \end{cases}$$

This means that the  $\text{which}$  function specifies the correct answers to the question which of the input processes terminates. It has many applications including one later in this paper.

A realizer of  $\text{which}$  can be defined from the projections  $\pi_i$  that get names of the components from a name of a pair via

$$F(\varphi)(n) := \begin{cases} \text{Some false} & \text{if } (\pi_0\varphi)(n) = \text{true} \\ \text{Some true} & \text{if } (\pi_1\varphi)(n) = \text{true} \\ \text{None} & \text{otherwise.} \end{cases}$$

The case distinction above is overlapping and we have to add that if more than one of the conditions are satisfied we choose the top-most option. This corresponds to a non-canonical choice and reordering the overlapping cases in the case distinction gives another valid realizer. Either of the realizers is clearly continuous and even computable and thus, the multivalued function  $\text{which}$  is computable and in particular continuous. However, both realizers return both names of  $\text{true}_{\mathbb{K}}$  and  $\text{false}_{\mathbb{K}}$  on input of two converging processes and switch between the return values depending on the names of these inputs. This is no coincidence, one may verify that no singlevalued choice function for  $\text{which}$  is continuously realizable.

The element  $\perp_{\mathbb{K}}$  of the Kleeneans stands for being undefined and the case distinction in the definition of  $\text{which}$  can be understood as extending a (partial) multivalued function in a canonical way to a total multivalued function. The use of such extensions is standard in more order-oriented models of computation [1]. In general, such an extension embodies a stricter specification than the non-extended version, as a realizer for the latter may behave arbitrarily on elements outside its domain, while a realizer for the former has to guarantee divergence. As the  $\text{which}$  function is computable, there is no difference in this case. Generally, there can only be a difference if the domain of the non-extended function is sufficiently complicated.

### 3.1 Operations on multifunctions and multivalued branching

Given  $\mathbf{f}: \mathbf{X} \rightrightarrows \mathbf{Y}$  and  $\mathbf{f}': \mathbf{X}' \rightrightarrows \mathbf{Y}'$  consider the multifunction  $\mathbf{f} \times \mathbf{f}': \mathbf{X} \times \mathbf{X}' \rightrightarrows \mathbf{Y} \times \mathbf{Y}'$  that on input of a pair returns the Cartesian product of the value sets, i.e.

$$(\mathbf{f} \times \mathbf{f}')(x, x') := \mathbf{f}(x) \times \mathbf{f}'(x').$$

As the Cartesian product is empty if one of the value sets is empty,  $\mathbf{f} \times \mathbf{f}'$  should be understood as the parallelization of  $\mathbf{f}$  and  $\mathbf{f}'$ . That is, if computable realizers of  $\mathbf{f}$  and  $\mathbf{f}'$  are given, a computable realizer for  $\mathbf{f} \times \mathbf{f}'$  can be specified by running the realizers for  $\mathbf{f}$  and  $\mathbf{f}'$  in parallel and returning something once both computations have come to an end.

To appropriately capture multivalued branching we need a similar operation for sums. Given  $\mathbf{f}: \mathbf{X} \rightrightarrows \mathbf{Y}$  and  $\mathbf{f}': \mathbf{X}' \rightrightarrows \mathbf{Y}'$  define a multifunction  $\mathbf{f} + \mathbf{f}': \mathbf{X} + \mathbf{X}' \rightrightarrows \mathbf{Y} + \mathbf{Y}'$  by

$$(\mathbf{f} + \mathbf{f}')(\mathit{p}) := \begin{cases} \{\text{inl } y \mid y \in \mathbf{f}(x)\} & \text{if } \mathit{p} = \text{inl } x \\ \{\text{inr } y' \mid y' \in \mathbf{f}'(x')\} & \text{if } \mathit{p} = \text{inr } x'. \end{cases}$$

Now, while  $\mathbf{f} \times \mathbf{f}'$  corresponds to parallel execution,  $\mathbf{f} + \mathbf{f}'$  corresponds to selective execution.

Next let us formulate branching over multivalued predicates. Consider the function  $\text{if}_{\mathbf{X}}: \mathbb{B} \times \mathbf{X} \rightarrow \mathbf{X} + \mathbf{X}$  defined by

$$\text{if}_{\mathbf{X}}(b, x) := \begin{cases} \text{inl } x & \text{if } b = \text{true} \\ \text{inr } x & \text{if } b = \text{false}. \end{cases}$$

Branching over the values of a function  $b: \mathbf{X} \rightarrow \mathbb{B}$  given  $f_0, f_1: \mathbf{X} \rightarrow \mathbf{Y}$  can be expressed using the  $\times$  and  $+$  operations and the  $\text{if}_{\mathbf{X}}$  function:

$$\text{if } b(x) \text{ then } f_1(x) \text{ else } f_0(x) = (\nabla \circ (f_1 + f_0) \circ \text{if}_{\mathbf{X}} \circ (b \times \text{id}) \circ \Delta)(x),$$

where  $\Delta(x) := (x, x)$  is the diagonal mapping and  $\nabla: \mathbf{Y} + \mathbf{Y} \rightarrow \mathbf{Y}$  is the backwards diagonal that returns  $y$  on both of the inputs  $\text{inl } y$  and  $\text{inr } y$ . Replacing the functions  $b, f_0$  and  $f_1$  by multifunctions is what we use as semantics for multivalued branching. The use of a sum reflects that only one of the if-statement branches should be evaluated. That is: if  $b(x) = \{\text{true}\}$  the eligible return-values are  $f_1(x)$  even if  $f_0(x)$  is empty, but if  $b(x) = \{\text{true}, \text{false}\}$  the eligible return-values of the if-statement are empty if either of  $f_0(x)$  and  $f_1(x)$  is empty and  $f_0(x) \cup f_1(x)$  otherwise. This is the behaviour one would expect from combining realizers.

#### 4 Representations for computation on the reals

The represented space  $\mathbb{R}_{\mathbb{Q}}$  from Section 2 is widely considered to provide the “correct” computability structure on the reals and is sometimes even used as a benchmark representation in works that reason about complexity in computable analysis. It provides an easy to understand question and answer structure that gives concrete meaning to the realizers. For the sake of automatically obtaining efficient algorithms carrying out a large number of arithmetic operations, on the other hand, other representations are superior. Such efficient representations should clearly reproduce the computability structure of  $\mathbb{R}_{\mathbb{Q}}$ .

Our goal is to provide a framework to define operations on real numbers without explicitly referring to implementation details while still allowing to replace the representations used and take advantage of some of their properties for improved performance. We therefore specify a set of operations that are convenient as building-blocks for higher-level operations. This can be seen as a computational axiomatization of the real numbers. Working relative to such an axiomatization allows to recompile the same algorithms for a new representation once these building-blocks, i.e. the axioms have been instantiated natively. Programs obtained this way can take better advantage of the details of the new representation than programs that just translate back and forth.

Other formal developments of real numbers such as the C-CoRn library use a constructive axiomatization. As the setting of our and prior work on real computation is fairly different, we chose to not directly reuse any of the constructive axiomatizations that can be found

in the literature [11]. Instead we used work from computable analysis such as [8, 10] and efficient non-verified software packages like iRRAM and AERN as guideline for choosing appropriate basic operations. We ended up requiring the following to be implemented:

- Arithmetic operations (addition, multiplication, subtraction and division),
- The efficient limit  $\lim_{\text{eff}} : \subseteq \mathbb{R}^\omega \rightarrow \mathbb{R}$ , that maps any sequence  $(x_i) \in \mathbb{R}^\omega$  that is **efficiently Cauchy**, i.e. such that for all  $i$  and  $j$ ,  $|x_i - x_j| \leq 2^{-i} + 2^{-j}$ , to its limit  $\lim(x_i)$ .
- The function  $\text{FtoR} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R}$ ,  $(m, e) \mapsto m \cdot 2^{-e}$  that embeds the dyadic rational numbers, or arbitrary-precision floating-point numbers, into  $\mathbb{R}$ .
- Rational approximation  $\text{approx} : \mathbb{R} \times \mathbb{Q} \rightrightarrows \mathbb{Q}$ , where  $\text{approx}(x, \varepsilon) := \{q \mid |x - q| \leq \varepsilon\}$ .
- The Kleenean comparison function  $<_{\mathbb{K}}$  of type  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{K}$ , defined from the Boolean comparison  $<$  on the reals by

$$x <_{\mathbb{K}} y := \begin{cases} (x < y)_{\mathbb{K}} & \text{if } x \neq y \\ \perp_{\mathbb{K}} & \text{otherwise.} \end{cases}$$

- A clean-up function that realizes the identity function  $\text{id} : \mathbb{R} \rightarrow \mathbb{R}$ . This function can always be instantiated with the identity function on the corresponding name spaces but in concrete cases it can be very useful as an optional performance enhancer that translates names to simpler names for the same object.

An equivalent formulation of the fourth item requires the availability of a translation to the rational representation from Example 1. The second and third item together are sufficient to define a translation in the other direction, so that any representation for which the above are instantiated is equivalent to the rational representation.

For the space  $\mathbb{R}_{\mathbb{Q}}$  from Example 1 we instantiated the above basic operations straightforwardly. This does not lead to satisfactory performance and there are several reasons for the inefficiency; one of these we addressed by providing a clean-up function: Iterated multiplication of rational numbers leads to huge numerators and denominators and this is exactly what happens if realizers are implemented using multiplication of rationals and then composed in a naive way. Efficiency can be recovered by replacing exact operations on rational numbers by rounded operations. Note that the direct use of rounded rational operations in the implementation of arithmetic operations would undermine the main advantage of the rational representation, namely, that the approximations have a nice mathematical structure. Instead, we round a rational name only when the clean-up operation is explicitly called.

#### 4.1 The interval reals and their arithmetic operations

There are more problems with the rational representation that make it difficult to optimize in applications. Approximations to the same real number may be required by different parts of an algorithm with different precision leading to extensive re-evaluation and the backwards propagation of errors requires building computation trees and results in blowup of time and space consumption if not done carefully. While these problems are in principle solvable, we decided to mostly use the rational representation for handling input and output and to translate to a representation based on sequences of intervals with dyadic endpoints [21]. Such a representation is commonly used in software packages for exact real arithmetic such as iRRAM [22]. The developers of C-CoRn made a similar switch in a fully constructive setting, also for performance reasons [12].

The **dyadic** numbers are the rational numbers of the form  $\frac{z}{2^n}$  for some  $z \in \mathbb{Z}$  and  $n \in \mathbb{N}$ . Let  $\mathbb{ID}$  be the set of all closed intervals with dyadic endpoints together with the infinite interval  $I_\infty := (-\infty, \infty)$ . For an interval  $I \in \mathbb{ID}$  let  $|I|$  denote the diameter of  $I$ ,

i.e.  $|[a, b]| := b - a$  and  $|I_\infty| = \infty$ . To define the represented space  $\mathbb{R}_{\mathbb{ID}}$  of Interval reals use  $\mathbf{Q}_{\mathbb{R}_{\mathbb{ID}}} := \mathbb{N}$ ,  $\mathbf{A}_{\mathbb{R}_{\mathbb{ID}}} := \mathbb{ID}$  and the representation  $\delta_{\mathbb{R}_{\mathbb{ID}}} : \subseteq \mathcal{B}_{\mathbb{R}_{\mathbb{ID}}} \rightarrow \mathbb{R}$  uniquely specified by

$$\delta_{\mathbb{R}_{\mathbb{ID}}}(I_n) = x \iff x \in \bigcap_{n \in \mathbb{N}} I_n \text{ and } \lim_{n \rightarrow \infty} |I_n| = 0.$$

That is, a sequence of intervals  $(I_n)_{n \in \mathbb{N}}$  is a name for  $x \in \mathbb{R}_{\mathbb{ID}}$  if  $x$  is contained in each interval and the diameter of the intervals approaches zero when  $n$  goes to infinity. In particular  $\bigcap_{n \in \mathbb{N}} I_n = \{x\}$  and we call the interval with index  $n$  the  $n$ -th approximation of  $x$ . We do not require the diameter to decrease monotonically as this would complicate operations and deteriorate performance.

In the formal development we made use of an existing formalization of interval arithmetic in COQ known as the COQ-interval library [20]. The library provides interval versions for many standard functions and in particular for arithmetic operations. For example, for any two intervals  $I, J \in \mathbb{ID}$  and any precision  $n \in \mathbb{N}$ , the COQ-interval function `add` returns an interval `add(n,I,J)` such that for all real numbers  $x, y$  with  $x \in I$  and  $y \in J$ ,  $x + y \in \text{add}(n, I, J)$ . The new endpoints are obtained by using arbitrary-precision floating-point operations with different rounding modes to compute the upper and lower interval bounds. The parameter  $n$  determines the bits used for the mantissa.

Using the COQ-interval functions, realizers of the arithmetic operations can be defined in a pointwise manner. The realizer for addition is e.g. defined as the function that maps  $(I_n)_{n \in \mathbb{N}}, (J_n)_{n \in \mathbb{N}}$  and a question  $n \in \mathbb{N}$  to `add(n, I_n, J_n)`. Here, and in other realizer definitions we round the  $n$ -th approximation to  $n$  mantissa digits to make the computational effort for different arithmetic operations on approximations with identical indices comparable. That these realizers return sequences of intervals each containing the correct result can be concluded from the inclusion property of the interval operations already proven in the interval library. Showing that the produced interval sequence converges requires bounds on the diameters that are not included in COQ-interval as they are not of particular interest for interval computation. We derive the error bounds from the theorems for the basic multiple precision floating-point operations from the FLOCQ library [7]. These operations use relative error bounds and we need bounds on the absolute error, which makes the proofs more complicated than one might first expect. The bounds usually depend not only on the diameter of the intervals but also on the values of the end-points.

## 4.2 The efficient limit operator and name cleanup

As compared to  $\mathbb{R}_{\mathbb{Q}}$ , an implementation of the limit operator on  $\mathbb{R}_{\mathbb{ID}}$  is more complicated. Recall that one has to transform a name of some efficiently Cauchy sequence  $(x_j) \subseteq \mathbb{R}$  to a name of its limit  $x$ . That is, given a sequence of sequences of intervals  $(I_{i,j})_{i,j \in \mathbb{N}}$  such that for each  $j \in \mathbb{N}$ ,  $x_j$  is contained in each  $I_{i,j}$  and  $|I_{i,j}| \rightarrow 0$  for  $i \rightarrow \infty$  the goal is to return a sequence  $(J_i)_{i \in \mathbb{N}}$  such that  $x \in J_i$  for all  $i$  and  $|J_i| \rightarrow 0$  for  $i \rightarrow \infty$ . The double-sequence  $(I_{i,j})$  can be thought of as an infinite matrix where each column contains a name, and intuitively it should be possible to find a name of the limit by traversing this matrix diagonally. However, it is at least necessary to slightly enlarge each interval to ensure that the limit is contained. But still after that, naively using the diagonal does not guarantee convergence. There are several strategies to search through the intervals and extract a name of the limit. In our implementation, we use a simple strategy known as *vertical search*. To get the  $n$ -th approximation, we choose the  $(n + 1)$ -st element of the sequence, do an unbounded search for an interval of size less than  $2^{-(n+1)}$  and extend it by  $2^{-(n+1)}$ . An advantage of this strategy is that it returns names with quickly converging intervals, resetting any precision loss incurred in other operations.

On concrete examples one quickly notices that computing a limit at low precisions tends to return useless results and yet takes a long time. This is because iterated use of arithmetic operations leads to intervals with large diameter and endpoints with big integer parts. We avoid this using the heuristic that the diameter of an interval should never be bigger than  $1/2$  so that at least the integer part of intermediate results is correct. This can be forced using a clean-up function that replaces any interval whose diameter is too large with the infinite interval. As the interval operations barely do any computation if one of the input intervals is infinite, this leads to a considerable speedup at low precision. Another cause for performance issues is the functional nature of names: Function values are not cached automatically leading to extensive reevaluation. As the questions are natural numbers in unary, there exists a simple solution: we internally replace the names by elements of a coinductive datatype of streams that are treated as lazy lists in evaluation.

## 5 A verified parametric square root algorithm

As a case study on how the basic operations can be used to define other operations on real numbers, let us study the example of the square root function in some detail. By the square root function we mean the partial function from reals to reals whose domain is  $[0, \infty)$  and whose return value on input of  $x \geq 0$  is  $\sqrt{x}$ . This function is a popular example as it being continuous but not analytic in 0 is a challenge in providing a good algorithm to compute it. We aim to recover this function in a compositional way from the basic operations listed in Section 4. A computable realizer for the square root function can then be extracted almost automatically by composing the realizers of the relevant basic operations independently from the exact implementation of the data-type of real numbers.

### 5.1 Square root approximation using Heron's method

A well known and efficient way to approximate the square root of a real number  $x$  is the Heron iteration inductively defined by  $x_0 := 1$  and  $x_{i+1} = \frac{1}{2} \left( x_i + \frac{x}{x_i} \right)$ . Let the function  $\text{heron} : \mathbb{R} \rightarrow \mathbb{R}^\omega$  be defined by  $\text{heron}(x)_i := x_{\lceil \log_2 i \rceil}$ . This function can be defined from our basic operations and returns an efficiently convergent sequence:

► **Lemma 5.**  $|\text{heron}(x)_i - \sqrt{x}| \leq 2^{-i}$ , whenever  $\frac{1}{4} \leq x \leq 2$ .

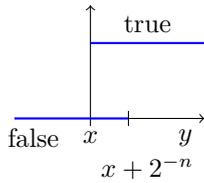
**Proof Sketch.** It is well-known that  $(x_i)$  converges quadratically to  $\sqrt{x}$  in the above interval. This means  $|x_i - \sqrt{x}| \leq 2^{-2^i}$  and thus heron returns an efficiently convergent sequence. ◀

Thus, the square root of some  $x \in [\frac{1}{4}, 2]$  can be approximated using heron and the efficient limit. We aim to extend the scope of this algorithm from the bounded interval  $[\frac{1}{4}, 2]$  to all of  $[0, \infty)$ . Our strategy is to handle 0 as a special case and scale strictly positive numbers to end up in the interval, apply the method above and then rescale the result appropriately. The following Lemma follows directly from Lemma 5.

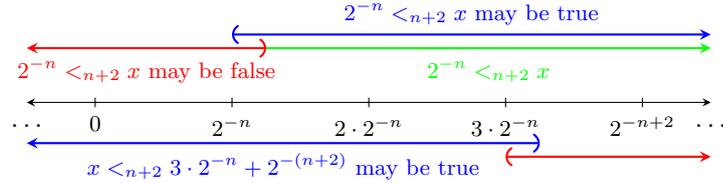
► **Lemma 6.** Let  $p \in \mathbb{Z}$  such that  $4^{-p}x \in [\frac{1}{4}, 2]$ , then  $|2^p \text{heron}(4^{-p}x)_{i+p} - \sqrt{x}| \leq 2^{-i}$ .

Let us call such a  $p$  a **scale** for  $x$ . Note that a scale exists if and only if  $x > 0$  and there always exists more than one possible choice in that case. Since  $\mathbb{Z}$  is discrete and  $\mathbb{R}$  connected, the semantics of an algorithm extracting an appropriate  $p$  from  $x$  are necessarily multivalued.

The treatment of the special case 0 requires branching. If  $x \leq 2^{-2^i}$  then 0 is already an approximation of the square root with error at most  $2^{-i}$ . Boolean-valued comparisons on the reals are discontinuous and therefore not computable. We may only use the Kleenean



■ **Figure 1**  $sc(n, x, y)$  plotted over  $y$  for fixed  $x$  and  $n$ .



■ **Figure 2**  $-n \in \text{mag}(x)$  if both inequality tests may be true. For  $x \in (0, 1)$  there is a number  $n \in \mathbb{N}$  such that both tests have true as the only valid value.

comparison  $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{K}$  that we included in our basic operations. Luckily, we do not need exact comparison but only need to know if either  $x > 0$  or  $x \leq 2^{-2i}$  and such a test can be implemented from the Kleenean comparison. To disregard the controlled divergence and define a total function in the end, we also go through multivaluedness.

For each of the previous two paragraphs let us develop some general purpose tools that may also be useful in other applications. For the branching needed around zero we use soft comparisons and for obtaining  $p$  we use a multivalued magnitude function, both of which we implement using the basic operations.

## 5.2 From Kleenean comparisons to soft comparison

Kleenean-valued comparisons are easy to implement but often inconvenient to use for implementation of total functions as they feature explicit divergence. A popular version of real number comparison trades off divergence for multivaluedness and is known as  $\varepsilon$ -test or soft comparison in numerics. For simplicity, we restrict to  $\varepsilon$  of the form  $2^{-n}$  and consider the multi-valued soft comparison  $sc: \mathbb{N} \times \mathbb{R} \times \mathbb{R} \rightrightarrows \mathbb{B}$  specified by

$$\text{true} \in sc(n, x, y) \Leftrightarrow x < y \quad \text{and} \quad \text{false} \in sc(n, x, y) \Leftrightarrow y < x + 2^{-n}. \tag{1}$$

This multifunction is total and properly multivalued as there is an interval of size  $2^{-n}$  where both cases overlap (see Figure 1).

Multivaluedness makes moving from prefix to infix notation more complicated. We fix the following conventions:  $x <_n y$  without any additions means  $sc(n, x, y) = \{\text{true}\}$ , for  $\text{true} \in sc(n, x, y)$  we state “ $x <_n y$  may be true” and for  $\text{false} \in sc(n, x, y)$  we write “ $x <_n y$  may be false”. This is illustrated by means of an example at the top of Figure 2. For functions  $f_0, f_1: \mathbb{R} \rightrightarrows \mathbb{R}$  expressions such as “if  $x <_n 0$  then  $f_1(x)$  else  $f_0(x)$ ” are meaningful as soft comparison is a multivalued predicate and branching works as explained in Section 3.1.

Soft comparison can be implemented using the Kleenean comparison and the which function from Example 4. We state this in the next Lemma. We use slightly sloppy notation as we identify the Booleans with a subspace of the Kleeneans, which in turn we consider elements of Sierpinski space by identifying  $\text{true}_{\mathbb{K}}$  with  $\top_{\mathbb{S}}$  and everything else with  $\perp_{\mathbb{S}}$ .

► **Lemma 7.** *Soft comparison can be expressed from the which function and  $<_{\mathbb{K}}$  via*

$$sc(n, x, y) = \text{which}(x <_{\mathbb{K}} y, y <_{\mathbb{K}} x + 2^{-n}).$$

**Proof.** No matter  $x$  and  $y$  we always have either  $x <_{\mathbb{K}} y = \text{true}_{\mathbb{K}}$  or  $y <_{\mathbb{K}} x + 2^{-n} = \text{true}_{\mathbb{K}}$ . Thus the which function on the above input always returns a Boolean. Further, true is a valid return value if and only if  $x < y$  and false is a valid return value if and only if  $y < x + 2^{-n}$ . ◀

### 5.3 The magnitude function for scaling

Recall that for computation of the value of the square root of a strictly positive real via rescaling, we needed to find an integer  $p$  such that  $x2^p$  is from a bounded interval and that such an integer cannot be found algorithmically without introducing multivaluedness. We thus implement the multifunction  $\text{mag}: \mathbb{R} \rightrightarrows \mathbb{Z}$  that extracts the magnitude of  $x$  in the sense that  $z \in \text{mag}(x) \Leftrightarrow 2^z < x < 2^{z+2}$ . Such a  $z$  exists whenever  $x > 0$ , i.e., the domain of magnitude are the positive real numbers.

Let us first argue that we may restrict to the case that  $0 < x < 1$ .

► **Lemma 8.** *The function  $\text{mag}$  can be recovered from its restriction to  $(0, 1)$  as*

$$\text{mag}(x) = \text{if } x <_1 2 \text{ then } \text{mag}|_{(0,1)}(x/2) + 1 \text{ else } -\text{mag}|_{(0,1)}(1/x) - 2.$$

**Proof.** In the first case  $x < 2$  and therefore  $x/2 \in (0, 1)$ . In the second case  $x > 3/2$  and therefore  $1/x \in (0, 1)$ . That the bounds are correct can be checked easily. ◀

Thus, assume  $0 < x < 1$  in the following.

► **Lemma 9.** *There always exists an  $n \in \mathbb{N}$  such that  $2^{-n} <_{n+2} x$  and  $x <_{n+2} 3 \cdot 2^{-n} + 2^{-(n+2)}$ , i.e. true is the only possible value for both conditions. Moreover, for any  $n \in \mathbb{N}$*

$$2^{-n} <_{n+2} x \text{ may be true} \wedge x <_{n+2} 3 \cdot 2^{-n} + 2^{-(n+2)} \text{ may be true} \quad \Longrightarrow \quad -n \in \text{mag}(x).$$

**Proof Sketch.**  $2^{-n} + 2^{-(n+2)} \leq x$  implies  $2^{-n} <_{n+2} x$  and  $x \leq 3 \cdot 2^{-n}$  implies  $x <_{n+2} 3 \cdot 2^{-n} + 2^{-(n+2)}$  (see Figure 2) and both these inequalities hold for instance for  $n = -\lceil \log_2(\frac{x}{3}) \rceil$ . Further, the two soft comparisons may only be true if  $2^{-n} < x < 3 \cdot 2^{-n} + 2^{-(n+2)} < 2^{-n+2}$  and thus  $-n \in \text{mag}(x)$ . ◀

In particular, a linear search for the first  $n$  such that the equation holds implemented using the realizers for the soft-comparison will always terminate and give a realizer for  $\text{mag}|_{(0,1)}$ , which in turn can be used to implement the full magnitude function via multivalued branching.

### 5.4 Defining the square root function

For any  $x > 0$  and  $m \in \text{mag}(x)$ ,  $\lceil \frac{m+1}{2} \rceil$  is a scale for  $x$  in the sense of Lemma 6. Thus, we finally have all tools necessary to define the square root function. We define an approximation function  $\text{sqapprox}: \mathbb{R} \rightrightarrows \mathbb{R}^\omega$  with domain  $[0, \infty)$  by

$$\text{sqapprox}(x)_i := \text{if } x <_{2i+1} 2^{-2i} \text{ then } 0 \text{ else } 2^p \text{heron}(4^{-p}x)_{i+p} \text{ where } p \text{ is a scale for } x.$$

Correctness is given by the following Lemma.

► **Lemma 10.**  $\lim_{\text{eff}} \circ \text{sqapprox}$  tightens the square root function.

**Proof.** It suffices to show that for each  $i \in \mathbb{N}$ ,  $\text{sqapprox}(x)_i$  is a  $2^{-i}$  approximation of  $\sqrt{x}$ . If  $x <_{2i+1} 2^{-2i}$  may be true then  $x < 2^{-2i}$  and 0 is a  $2^{-i}$  approximation of  $\sqrt{x}$ . If  $x <_{2i+1} 2^{-2i}$  may be false then  $2^{-(2i+1)} \leq x$  and thus  $x \in \text{dom}(\text{mag})$  and we can apply Lemma 6. ◀

This means that any realizer for  $\lim_{\text{eff}} \circ \text{sqapprox}$  is also a realizer for the square root function and can thus be defined only by using realizers for basic operations.

## 6 Implementation

All results of this paper have been formally verified in the COQ proof assistant. The implementation is part of the INCONE library. It is in the development branch and will be featured in a future release. An overview of the library and instructions on how to get started can be found in [24]. The content of the current work can be found in a folder for examples about real numbers in the development branch of the library [23]. The real number structure from Section 4, the treatment of interval reals and the interval representation are each given their own files in that folder. The error bound estimates for operations from the COQ-interval library needed for the interval representation have been exported to a separate file so that the file sizes remain manageable. The content of Section 5 is separated into a file for the soft comparison, one for the magnitude function and finally one where the square root function is implemented. The finite spaces from Section 3 have been integrated into INCONE and can be found in the folder for constructions on continuity spaces under the name “hyperspaces”.

Our development uses a fairly small set of axioms, namely those used in the axiomatic formalization of the reals, the law of excluded middle, functional extensionality and some choice principles. The reasoning is usually divided into two parts, where the first is coming by with mathematics and the second part is to define realizers and prove them correct. We carefully define the realizers such that they do not rely on the non-constructive axioms of the reals and actually correspond to executable programs.

As a concrete example, let us consider some parts of the formalization of the square root algorithm from Section 5. While the more difficult part and most of the content of the `sqrt.v` file constitutes the extension to the whole real line, for simplicity we here only consider the restriction to the interval  $[\frac{1}{4}, 2]$  where the Heron method converges quadratically.

The COQ standard library already defines a function `sqrt` for the square root built on the axiomatization of the reals and proves some of its properties. Assume we have fixed some representation of the reals and denote the spaces of questions and answers by  $\mathbf{Q}$  and  $\mathbf{A}$ , respectively. An algorithm implementing the square root function thus takes and returns elements of  $\mathbf{A}^{\mathbf{Q}}$ . In a first approximation such an algorithm may be represented by a COQ function of type `sqrt_rlzr: (Q -> A) -> (Q -> A)`. For this function to actually correspond to an algorithm, its definition should not involve incomputable axioms. The correctness of the algorithm is guaranteed by a specification Lemma of the form:

**Lemma** `sqrt_rlzr_spec: sqrt_rlzr \realizes sqrt.`

The notation `\realizes` is part of the INCONE library and means that `sqrt_rlzr( $\varphi$ )` is name of `sqrt(x)` whenever  $\varphi$  is a name of  $x$ . INCONE defines several such notations making the formal statements look very similar to the informal mathematical statements.

Unfortunately, the situation is usually more complicated as the realizer may need to be partial. Some algorithm can diverge if the input is not a valid name of a real number. In COQ, all functions are total but partial functions can be modeled using dependent types. A partial function takes as input a pair consisting of the actual input and a proof that this input is contained in its domain. Note that COQ’s `sqrt` function itself is not partial. For the restriction, as the realized function does not carry computational information, we take a different approach to partiality here and instead of using the dependent type system we move to a relational specification right away. That is, we replace the function `sqrt` by its induced multifunction `F2MF sqrt` which can be restricted by adding a domain condition. Finally we may bundle the realizing function with its correctness proof so that result to be found in INCONE actually takes the following form:

```

Lemma sqrt_rlzr_exists :
  {f : partial_function | f \solves (F2MF sqrt)|_[/4,2]}.

```

The partial function itself can be retrieved by `(sval sqrt_rlzr_exists)` and its correctness proof by `(svalP sqrt_rlzr_exists)`. The function definition can be extracted to Haskell code and then be executed. The terms usually fail to reduce internally due to the use of non-computational real numbers in the specification part that entangled with the definitional part through the use of partial functions and sigma types.

For broad applicability we do not only work with a specific representation but define a structure of `computable_reals` that can be instantiated with different representations. This structure closely resembles the informal description in Section 4 and serves as an intermediate level for real number operations. It contains a representation for the reals and partial functions realizing the basic operations together with correctness proofs. Other operations can be defined by composition, product, sums and branching on the basic operations. For instance, the square root function in our implementation has a parameter `Rc` of type `computable_reals` and returns for each instance of this structure an executable program. The actual algorithm is based on Heron's method and closely follows the outline in Section 5. Let us list the definition and specification Lemma of the `sqrt_approx` function from the paper as an example:

```

Fixpoint sqrt_approx x0 n x :=
  match n with
  | 0 => x0
  | S n' => let x' := sqrt_approx x0 n' x in (x' + x / x') / 2
  end.

```

```

Lemma sqrt_approx_correct x n:
  /4 <= x <= 2 -> Rabs (sqrt_approx 1 n x - sqrt x) <= /2^(2^n).

```

The INCONE library equips the space of infinite sequences with the structure of a represented space again. An algorithm to find the limit of an efficiently convergent sequence as operation `lim_eff: Rcω -> Rc` is part of the `computable_reals` structure. As outlined in Section 5 one can use the iteration above to obtain a function `heron: Rc -> Rcω` that returns an efficiently convergent sequence for certain real numbers. The composition of `heron` with the limit operator returns the `sqrt` function:

```

Lemma sqrt_as_lim :
  (lim_eff \o heron) \tightens (F2MF sqrt)|_[/4,2]

```

Once this specification is available, a realizer of the right hand side can be obtained from the realizers of the operations on the left hand side by compositionality. The realizer of the `heron` function is obtained by piecing together the realizers for the arithmetic operations.

## 6.1 Executability and code extraction

As the definitions of the domains of the partial functions we use as realizers involve the non-computational real numbers, the corresponding functions can not be executed COQ internally using term reduction. These problems can be worked around for instance by using a fuel-based approach [18], but in our current implementation this method leads to considerably worse running times and we therefore refrain from giving details here. An additional drawback is that extra information about the realizers of the basic operations is needed and additional work is necessary for propagating this information through operations such as composition and taking fix points.

As the main purpose of our implementation is not to do computation inside COQ but to provide an easy to use interface for developers in exact real computation frameworks to define and verify their algorithms, we focus on using COQ's code extraction features to generate efficient code instead of direct execution inside of COQ. We hope that the extracted programs can be integrated into other developments for parts where particularly strong correctness guarantees are needed.

COQ's code extraction feature can be used to generate executable programs. However, the performance of the extracted programs depends on how the extraction is done exactly. For instance, if the basic operations on integers are translated from their COQ implementation that is targeted towards simple proofs instead of efficiency, the performance will suffer. It is possible to instruct COQ to extract these operations to native implementations in Haskell instead. We have extracted all arithmetic operations and comparisons on integer types such as `Z`, `nat` or `positive` to the corresponding operations on arbitrary-sized integers in Haskell. Some other operations such as shifting and taking integer logarithms that turned out to be particularly slow were also replaced by more efficient implementations available in Haskell. Of course, these Haskell operations are not formally verified and each modification increases the size of the trusted core and the risk for errors in the final program. As the set of operations we trust for the extraction is quite small, we believe this risk to be manageable.

The replacement of functions by streams discussed in Section 4.2 can either be done directly in COQ or in Haskell by adding some instructions for the extraction. The replacement in Haskell performed slightly better in experiments and we used it as the default option.

## 7 Conclusion and Future work

On paper the approach of computable analysis is very much in accordance with the spirit of COQ. In principle it should be possible to parameterize the theories over an abstract type so that the classical treatment of real numbers and similar structures is hidden in the propositional layer. In practice there are a lot of additional hurdles in maintaining a clean mathematical presentation, executability and reuseability of existing work. Much of the existent infrastructure for computation on the reals such as the `Flocq` and `Interval` libraries are specified against the classical axiomatization of real numbers from the standard library. This axiomatization of the real numbers states classical properties, such as decidability of equality, as global facts and makes maintaining executability challenging.

Currently, we have only implemented a few basic operations on real numbers, mostly to demonstrate that our framework indeed can be used for efficient computation. Adding further operations such as trigonometric functions should not be too difficult. An interesting direction for future work is to extend the computation on real numbers to operators on real functions such as integration and ODE solving. The tools contained in the `INCONE` library can already be used to automatically generate a representation for real functions from a representation for real numbers. Ideas from real number complexity theory [16, 15] suggest that the use of specialized representations over this generic function representation might yield even better results.

---

### References

- 1 S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, Oxford, 1994.

- 2 Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *Journal of Formalized Reasoning*, 11(1):43–76, 2018. doi:10.6092/issn.1972-5787/8124.
- 3 Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *In: Proceedings of the International Symposium on Mathematical Theory of Networks and Systems.*, 2006.
- 4 Merrie Bergmann. *An introduction to many-valued and fuzzy logic: semantics, algebras, and derivation systems*. Cambridge University Press, 2008.
- 5 Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015.
- 6 Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016. URL: <http://hal.inria.fr/hal-00806920>.
- 7 Sylvie Boldo and Guillaume Melquiond. Flocq: A unified library for proving floating-point algorithms in coq. In *2011 IEEE 20th Symposium on Computer Arithmetic*, pages 243–252. IEEE, 2011.
- 8 Vasco Brattka and Peter Hertling. Feasible real random access machines. *J. Complexity*, 14(4):490–526, 1998. doi:10.1006/jcom.1998.0488.
- 9 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A Tutorial on Computable Analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 425–491. Springer, 2008.
- 10 Franz Brauße, Pieter Collins, Johannes Kanig, SunYoung Kim, Michal Konečný, Gyesik Lee, Norbert Müller, Eike Neumann, Sewon Park, Norbert Preining, et al. Semantics, logic, and verification of "exact real computation". *arXiv preprint arXiv:1608.05787*, 2016.
- 11 Alberto Ciaffaglione and Pietro Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science*, 351(1):39–51, 2006. Real Numbers and Computers. doi:10.1016/j.tcs.2005.09.061.
- 12 Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.
- 13 Martín Hötzel Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, 1996.
- 14 Akitoshi Kawamura. *Computational complexity in analysis and geometry*. University of Toronto, 2011.
- 15 Akitoshi Kawamura, Norbert Th. Müller, Carsten Rösnick, and Martin Ziegler. Computational Benefit of Smoothness. *Journal of Complexity*, 2015. doi:10.1016/j.jco.2015.05.001.
- 16 Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ODE solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.
- 17 Michal Konečný. AERN-Real: Arbitrary-precision interval arithmetic for approximating exact real numbers, 2008.
- 18 Michal Konečný, Florian Steinberg, and Holger Thies. Continuous and Monotone Machines. In Javier Esparza and Daniel Král, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.56.
- 19 Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical computer science*, 38:35–53, 1985.
- 20 Guillaume Melquiond. Proving bounds on real-valued functions with computations. In *International Joint Conference on Automated Reasoning*, pages 2–17. Springer, 2008.

- 21 R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.
- 22 Norbert Th. Müller. The iRRAM: Exact arithmetic in C++. In *Computability and complexity in analysis. 4th international workshop, CCA 2000. Swansea, GB, September 17–19, 2000. Selected papers*, pages 222–252. Berlin: Springer, 2001.
- 23 Florian Steinberg. The INCONE library. <https://github.com/FlorianSteinberg/incone>, 2019. release v1.0.
- 24 Florian Steinberg, Laurent Thery, and Holger Thies. Computable analysis and notions of continuity in coq. *arXiv preprint arXiv:1904.13203*, 2019.
- 25 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin/Heidelberg, 2000.

## A Experimental results

To show that our implementation indeed gives a feasible implementation of exact real computation we did a small experimental study where we compared the running times to approximate some simple functions using our implementation to an implementation in the C-CoRn library and non-verified implementations using exact real arithmetic packages. While the experiments show that our implementation is not yet optimal, the difference in running time was only by a small factor and we think that it could be further reduced by optimizing our representation.

For all experiments we extracted Haskell code from the specification in COQ (version 8.9.0) using the code extraction mechanism. Apart from the simple optimizations for the code extraction mentioned above we did not do any additional changes to increase performance. In particular we did not change the extracted code except for adding a few includes of standard Haskell libraries in the beginning of the file. The Haskell code was compiled with GHC version 8.8.1 and profiling options turned on. The running times were taken from the total time written in the Time and Allocation Profiling Report generated by Haskell. All experiments were done on a Macbook Pro 2015 model with 16 GB RAM and 2.2 GHz Intel Core i7 processor. We tried the experiments with both the rational and interval representation for real numbers, however as expected the (non-optimized) rational representation performed very poorly and we thus focus on the results for the interval representation.

We also compared the running time to computing the same problem with the C-CoRn library (using the same code extraction techniques) and a (non-verified) C++ implementation using the iRRAM framework. These comparisons have to be taken with a grain of salt as many details of the implementations differ. For instance, our implementation outputs the result as a rational number giving numerator and denominator while iRRAM and C-CoRn output decimal approximations.

The first experiment is to compute iterations of the logistic map  $x_{n+1} = rx_n(1 - x_n)$  for  $x_0 = 0.5$  and  $r = 3.75$ . The logistic map is often used as a benchmark problem in exact real computation as it exhibits chaotic behavior, i.e., a slight change in the initial condition leads to completely different values at later iterations. In particular, computations using standard floating-point methods quickly diverge from the correct solution. While it may be argued that computing the exact values is not of any practical relevance, it is a popular example for where floating point computations fail completely, while exact methods can quickly produce correct results.

In this experiment we output an approximation of the result after several iterations of the logistic map with error less than  $10^{-1000}$  (i.e. approximately 1000 decimal digits). In our experiments our implementation performed quite well (see Table 3a) and was only a

$N$	INCONE	iRRAM
100	0.02	0
500	0.1	0.01
1000	0.18	0.01
5000	0.94	0.27
10000	3.03	0.83
20000	12.02	4.32
50000	67.23	38.4

$n$	$\sqrt{5}$	$\sqrt{\frac{5}{32}}$
10	0.12	0.08
100	0.33	0.23
500	1.05	0.64
1000	1.78	1.12
2000	2.74	1.76
5000	6.33	3.61
10000	8.51	5.13
50000	18.02	10.33
100000	27.93	15.98
500000	91.1	48.72

(a) Approximating 1000 digits of the  $N$ -th iteration of the logistic map.

(b) Computing  $n$  digits of the square root.

■ **Figure 3** Running times (seconds) for the different experiments.

factor 2 – 5 slower than the iRRAM implementation. A straightforward implementation in C-CoRn did not give good results as evaluating  $x_n$  twice in the iteration rule leads to exponential growth and therefore already computing more than a few iterations takes a very long time. However, this is probably just due to our naive implementation and it might be possible to do a more clever implementation in C-CoRn that caches the intermediate values.

Our second experiment was to compute some square roots, i.e., compute the square root of a given rational number and output an approximation with a certain error bound. We give the results for  $\sqrt{5}$  and  $\sqrt{\frac{5}{32}}$  as representatives for numbers that are scaled down resp. up in our algorithm. Other numbers performed mostly similarly, however as computing the magnitude uses a linear search for very large numbers the running time gets significantly worse. Here, while our algorithm is still usable, its performance was far worse than both the iRRAM and C-CoRn versions. For example iRRAM could still compute 500000 digits in less than 0.01 seconds. The C-CoRn version was nearly as fast as the iRRAM version for up to 10000 digits. For higher precision it got significantly slower and for 500000 digits even performed worse than our implementation. The performance log shows that this is not a bug in C-CoRn but due to some integer operation being extracted to a sub-optimal implementation. As C-CoRn is made for execution inside of COQ and not optimized for Haskell code extraction, it is quite hard to compare these numbers.

Our implementation has similar issues when using the interval library. The COQ interval library is built for fast execution inside of COQ, however that makes the extracted code quite complicated and many operations could be implemented much more efficiently in Haskell. Moving to a simpler implementation of interval arithmetic should therefore lead to a drastic improvement.

As the performance hugely depends on factors that have mostly to do with code extraction, it is questionable how valuable a thorough performance comparison of the different frameworks is. We think the main take-away message from this experimental study should be that while possibly not as fast as some of the alternatives, our simple implementation still performs reasonably well and can be used to compute approximations up to very high precision.

# Perspective Games with Notifications

**Orna Kupferman**

School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel  
orna@cs.huji.ac.il

**Noam Shenwald**

School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel  
noam.shenwald@mail.huji.ac.il

---

## Abstract

A reactive system has to satisfy its specification in all environments. Accordingly, design of correct reactive systems corresponds to the synthesis of winning strategies in games that model the interaction between the system and its environment. The game is played on a graph whose vertices are partitioned among the players. The players jointly generate a path in the graph, with each player deciding the successor vertex whenever the path reaches a vertex she owns. The objective of the system player is to force the computation induced by the generated infinite path to satisfy a given specification. The traditional way of modelling uncertainty in such games is observation-based. There, uncertainty is longitudinal: the players partially observe all vertices in the history. Recently, researchers introduced *perspective games*, where uncertainty is transverse: players fully observe the vertices they own and have no information about the behavior of the computation between visits in such vertices. We introduce and study *perspective games with notifications*: uncertainty is still transverse, yet a player may be notified about events that happen between visits in vertices she owns. We distinguish between structural notifications, for example about visits in some vertices, and behavioral notifications, for example about the computation exhibiting a certain behavior. We study the theoretic properties of perspective games with notifications, and the problem of deciding whether a player has a winning perspective strategy. Such a strategy depends only on the visible history, which consists of both visits in vertices the player owns and notifications during visits in other vertices. We show that the problem is EXPTIME-complete for objectives given by a deterministic or universal parity automaton over an alphabet that labels the vertices of the game, and notifications given by a deterministic satellite, and is 2EXPTIME-complete for LTL objectives. In all cases, the complexity in the size of the graph and the satellite is polynomial – exponentially easier than games with observation-based partial visibility. We also analyze the complexity of the problem for richer types of satellites.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory; Theory of computation → Logic and verification

**Keywords and phrases** Games, Incomplete Information, Automata

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.51

**Related Version** <https://www.cs.huji.ac.il/~ornak/publications/fsttcs20.pdf>.

## 1 Introduction

A reactive system has to satisfy its specification in all environments. Accordingly, design of correct reactive systems corresponds to the synthesis of a winning strategy for the system in a game that model the interaction between the system and its environment. The game is played on a graph whose vertices correspond to configurations along the interaction. We study here settings in which each configuration is controlled by either the system or its environment. Thus, the set of vertices is partitioned between the players, and the game is *turn-based*: starting from an initial vertex, the players jointly generate a *play*, namely a path in the graph, with each player deciding the successor vertex when the play reaches a vertex she controls. Each vertex is labeled by an assignment to a set  $AP$  of atomic propositions –



© Orna Kupferman and Noam Shenwald;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 51; pp. 51:1–51:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

these with respect to which the system is defined. The objective of the system is given by a language  $L \subseteq (2^{AP})^\omega$ , and it wins if the computation induced by the generated play, namely the word that labels its vertices, is in  $L$  [14, 4].

A *strategy* for a player directs her how to continue a play that reaches her vertices. We consider *deterministic* strategies, which choose a successor vertex. In games with *full visibility*, strategies may depend on the full history of the play. In games with *partial visibility*, strategies depend only on visible components of the history [16]. A well studied model of partial visibility is *observation based* [9, 6, 5, 2]. There, a player does not see the vertices of the game and can only observe the assignments to a subset of the atomic propositions. Accordingly, strategies cannot distinguish between different plays in which the observable atomic propositions behave in the same manner. Recently, [8] introduced *perspective games*. There, the visibility of each player is restricted to her vertices. Accordingly, a perspective strategy for a player cannot distinguish among histories that differ in visits to vertices owned by other players. As detailed in [8], the perspective model corresponds to switched systems and component-based software systems [1, 11, 12, 13].

Note that visibility and lack of visibility in the observation-based model are *longitudinal* – players observe all vertices, but partially. On the other hand, in the perspective model, players have full visibility on the parts of the system they control, and no visibility (in particular, even no information on the number of transitions taken) on the parts they do not control. Thus, visibility and lack of visibility are *transverse* – some vertices the players do not see at all, and some they fully see. For a comparison of perspective games with related visibility models (in particular, games with partial visibility in an asynchronous setting [15], switched systems [7], and control-flow composition in software and web service systems [12]), see [8].

In many settings, players indeed cannot observe the evolution of the computation in parts of the system they do not control, yet they may have information about events that happen during these parts. For example, if the system is synchronous with a global clock, then all players know the length of the invisible parts of the computation. Likewise, visits in some vertices of the other players may be observable, for example in a communication network in which all companies observe routers that belong to an authority and can detect visits to routers that leave a stamp. Finally, behaviors may be visible too, like an airplane that flies high, or a robot that enters a zone that causes an alarm to be activated. In this paper we introduce and study *perspective games with notifications*, which model such settings.

Formally, perspective games with notifications include, in addition to the game graph and the winning condition, an *information satellite*: a finite state machine that is executed in parallel with the game and may notify the players about events it monitors. We distinguish between *structural* satellites, which monitor the generated play, and *behavioral* satellites, which monitor the generated computation. Examples to structural satellites include ones that notify the players about visits in designated sets of states, transitions among regions in the system, say calls and returns in software systems, traversal of loops, etc. Another useful structural satellite notifies the players about the assignment to a subset of the atomic propositions. Note that such a satellite combines the transverse visibility of perspective games with the longitudinal visibility in observation-based games. A typical behavioral satellite is associated with a regular language  $R \subseteq (2^{AP})^*$ . The satellite may notify the players whenever the computation induced by the play is in  $R$  (termed a *single-track* satellite), or whenever a suffix of the computation is in  $R$  (termed a *multi-track* satellite). The language  $R$  may vary from simple propositional assertion over  $AP$ , to rich finite on-going behaviors. Note that even very simple satellites may be very useful. For example, when  $R = (2^{AP})^*$ , the satellite acts as a clock, notifying the players about the length of the invisible parts of the computation.

We start by studying some theoretical aspects of perspective games with notifications. We consider two-player games with a winning condition  $L \subseteq (2^{AP})^\omega$  such that PLAYER 1 aims for a play whose computation is in  $L$ , and PLAYER 2 aims for a play whose computation is not in  $L$ . Unsurprisingly, the basic features of the game are inherited from the model without notifications. In particular, perspective games with notifications are not determined. Thus, there are games in which PLAYER 1 does not have a perspective strategy that forces the generated computation to satisfy  $L$  nor PLAYER 2 has a perspective strategy that forces the generated computation not to satisfy  $L$ . Also, the restriction to a perspective strategy (as opposed to one that fully observes the computation) makes a difference only for one of the players. Thus, if PLAYER 1 has a strategy to win against all perspective strategies of PLAYER 2, she also has a perspective strategy to win against all strategies of PLAYER 2.

The prime problem when reasoning about games is to decide whether a player has a winning strategy. Here the differences between perspective games and other models of partial visibility become significant: handling of observation-based partial visibility typically involves some subset-construction-like transformation of the game graph into a game graph of exponential size with full visibility. Accordingly, deciding of observation-based partial-visibility games is EXPTIME-complete in the graph [2, 6, 5, 3]. In perspective games, one can avoid this exponential blow-up in the size of the graph and trade it with an exponential blow-up in the (typically much smaller) winning condition [8].

Our main technical contribution is an extension of these good news to perspective games with notifications, and a study of the complexity in terms of the satellite. The solution in [8] is based on the definition of a tree automaton for winning strategies. The extension to a model with notifications is not easy, as the type of strategies is different. Let  $V_1$  denote the set of vertices that PLAYER 1 controls. With no notifications, a strategy for PLAYER 1 is a function  $f : V_1^* \rightarrow V$ , mapping each visible history to a successor vertex. With notifications, the visible histories of PLAYER 1 consist not only of vertices in  $V_1$  but refer also to a set  $I$  of notifications that PLAYER 1 may receive from the satellite. Moreover, histories that end in a notification in  $I$  correspond to vertices in the game in which PLAYER 1 do not have control. Accordingly, the outcome of the strategy in them is not important, yet they should still be taken into account. We are still able to define a tree automaton for winning strategies. Essentially, the tree automaton follows both the satellite and the automaton for the winning condition, where a tree that encodes a strategy includes branches not only for vertices in  $V_1$  but also branches for notifications in  $I$ . We analyze the complexity of our algorithm for winning conditions given by deterministic or universal co-Büchi or parity automata, as well as by LTL formulas, and show that the problem is EXPTIME-complete for all above types of automata and is 2EXPTIME-complete for LTL. In all cases, the complexity in terms of the graph and the satellite is polynomial.

While EXPTIME-hardness follows immediately from the setting with no notifications [8], we analyse the complexity also in terms of the satellite. Recall that given a finite language  $R \subseteq (2^{AP})^*$ , a satellite may be single-track, notifying about computations in  $R$ , or multi-track, notifying about computations in  $(2^{AP})^* \cdot R$ . We examine four cases, depending on whether the satellite is single- or multi-track and whether  $R$  is given by a deterministic or nondeterministic automaton. For deterministic single-track satellites, the complexity of deciding whether PLAYER 1 wins is polynomial. In the other three cases, a naive construction of a satellite requires determinization and involves an exponential blow-up. Note that this applies also to the case where  $R$  is given by a deterministic automaton yet the satellite is multi-track, and thus has to follow all suffixes. We show that this blow up is unavoidable. Thus, deciding whether PLAYER 1 wins is EXPTIME-hard even when the winning condition,

which is the source for the exponential complexity in the setting with no notifications, is fixed. On the positive side, we show that many interesting cases need a fixed-size satellite, or a satellite whose state space can be merged with that of the game.

## 2 Preliminaries

### 2.1 Perspective games

A *game graph* is a tuple  $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$ , where  $AP$  is a finite set of atomic propositions,  $V_1$  and  $V_2$  are disjoint sets of vertices, owned by PLAYER 1 and PLAYER 2, respectively, and we let  $V = V_1 \cup V_2$ . Then,  $v_0 \in V_1$  is an initial vertex, which we assume to be owned by PLAYER 1, and  $E \subseteq V \times V$  is a total edge relation, thus for every  $v \in V$  there is  $u \in V$  such that  $\langle v, u \rangle \in E$ . The function  $\tau : V \rightarrow 2^{AP}$  maps each vertex to a set of atomic propositions that hold in it. The size  $|G|$  of  $G$  is  $|E|$ , namely the number of edges in it.

In a beginning of a play in the game, a token is placed on  $v_0$ . Then, in each turn, the player that owns the vertex that hosts the token chooses a successor vertex and move there the token. A *play*  $\rho = v_0, v_1, \dots$  in  $G$ , is an infinite path in  $G$  that starts in  $v_0$ ; thus for all  $i \geq 0$  we have that  $\langle v_i, v_{i+1} \rangle \in E$ . The play  $\rho$  induces a *computation*  $\tau(\rho) = \tau(v_0), \tau(v_1), \dots \in (2^{AP})^\omega$ .

A *game* is a pair  $\mathcal{G} = \langle G, L \rangle$ , where  $G$  is a game graph, and  $L \subseteq (2^{AP})^\omega$  is a *behavioral winning condition*, namely an  $\omega$ -regular language over the atomic propositions, given by an LTL formula or an automaton. Intuitively, PLAYER 1 aims for a play whose computation is in  $L$ , while PLAYER 2 aims for a play whose computation is in  $\text{comp}(L) = (2^{AP})^\omega \setminus L$ .

Let  $\text{Prefs}(G)$  be the set of nonempty prefixes of plays in  $G$ . For a sequence  $\rho = v_0, \dots, v_n$  of vertices, let  $\text{Last}(\rho) = v_n$ . For  $j \in \{1, 2\}$ , let  $\text{Prefs}_j(G) = \{\rho \in \text{Prefs}(G) : \text{Last}(\rho) \in V_j\}$ . In games with full visibility, the players have a full view of the generated play. Accordingly, a strategy for PLAYER  $j$  maps  $\text{Prefs}_j(G)$  to vertices in  $V$  in a way that respects  $E$ . In *perspective games* [8], PLAYER  $j$  can view only visits to  $V_j$ . Accordingly, strategies are defined as follows. For a prefix  $\rho = v_0, \dots, v_i \in \text{Prefs}(G)$ , and  $j \in \{1, 2\}$ , the *perspective of player  $j$  on  $\rho$* , denoted  $\text{Persp}_j(\rho)$ , is the restriction of  $\rho$  to vertices in  $V_j$ . We denote the perspectives of player  $j$  on prefixes in  $\text{Prefs}_j(G)$  by  $\text{PPrefs}_j(G)$ , namely  $\text{PPrefs}_j(G) = \{\text{Persp}_j(\rho) : \rho \in \text{Prefs}_j(G)\}$ . Note that  $\text{PPrefs}_j(G) \subseteq V_j^*$ . A *perspective strategy* for player  $j$ , is then a function  $f_j : \text{PPrefs}_j(G) \rightarrow V$  such that for all  $\rho \in \text{PPrefs}_j(G)$ , we have that  $\langle \text{Last}(\rho), f_j(\rho) \rangle \in E$ . That is, a perspective strategy for player  $j$  maps her perspective of prefixes of plays that end in a vertex  $v \in V_j$  to a successor of  $v$ .

The *outcome* of P-strategies  $f_1$  and  $f_2$  for PLAYER 1 and PLAYER 2, respectively, is the play obtained when the players follow their P-strategies. Formally,  $\text{Outcome}(f_1, f_2) = v_0, v_1, \dots$  is such that for all  $i \geq 0$  and  $j \in \{1, 2\}$ , if  $v_i \in V_j$ , then  $v_{i+1} = f_j(\text{Persp}_j(v_0, \dots, v_i))$ .

We use  $F$  and  $P$  to indicate the visibility type of strategies, namely whether they are full ( $F$ ) or perspective ( $P$ ). Consider a game  $\mathcal{G} = \langle G, L \rangle$ . For  $\alpha, \beta \in \{F, P\}$ , we say that PLAYER 1 ( $\alpha, \beta$ )-wins  $\mathcal{G}$  if there is an  $\alpha$ -strategy  $f_1$  for PLAYER 1 such that for every  $\beta$ -strategy  $f_2$  for PLAYER 2, we have that  $\tau(\text{Outcome}(f_1, f_2)) \in L$ . Similarly, PLAYER 2 ( $\alpha, \beta$ )-wins  $\mathcal{G}$  if there is an  $\alpha$ -strategy  $f_2$  for PLAYER 2 such that for every  $\beta$ -strategy  $f_1$  for PLAYER 1, we have that  $\tau(\text{Outcome}(f_1, f_2)) \notin L$ .

### 2.2 Automata

Given a set  $D$  of directions, a *D-tree* is a set  $T \subseteq D^*$  such that if  $x \cdot c \in T$ , where  $x \in D^*$  and  $c \in D$ , then also  $x \in T$ . The elements of  $T$  are called *nodes*, and the empty word  $\varepsilon$  is the *root* of  $T$ . For every  $x \in T$ , the nodes  $x \cdot c$ , for  $c \in D$ , are the *successors* of  $x$ . A *path*  $\pi$

of a tree  $T$  is a set  $\pi \subseteq T$  such that  $\varepsilon \in \pi$  and for every  $x \in \pi$ , either  $x$  is a leaf or there exists a unique  $c \in D$  such that  $x \cdot c \in \pi$ . Given an alphabet  $\Sigma$ , a  $\Sigma$ -labeled  $D$ -tree is a pair  $\langle T, \tau \rangle$  where  $T$  is a tree and  $\tau : T \rightarrow \Sigma$  maps each node of  $T$  to a letter in  $\Sigma$ .

For a set  $X$ , let  $\mathcal{B}^+(X)$  be the set of positive Boolean formulas over  $X$  (i.e., Boolean formulas built from elements in  $X$  using  $\wedge$  and  $\vee$ ), where we also allow the formulas **true** and **false**. For a set  $Y \subseteq X$  and a formula  $\theta \in \mathcal{B}^+(X)$ , we say that  $Y$  *satisfies*  $\theta$  iff assigning **true** to elements in  $Y$  and assigning **false** to elements in  $X \setminus Y$  makes  $\theta$  true. An *alternating tree automaton* is  $\mathcal{A} = \langle \Sigma, D, Q, q_{in}, \delta, \alpha \rangle$ , where  $\Sigma$  is the input alphabet,  $D$  is a set of directions,  $Q$  is a finite set of states,  $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$  is a transition function,  $q_{in} \in Q$  is an initial state, and  $\alpha$  is an acceptance condition. We consider here the Büchi, co-Büchi, and parity acceptance conditions. For a state  $q \in Q$ , we use  $\mathcal{A}^q$  to denote the automaton obtained from  $\mathcal{A}$  by setting the initial state to be  $q$ . The *size* of  $\mathcal{A}$ , denoted  $|\mathcal{A}|$ , is the sum of lengths of formulas that appear in  $\delta$ .

The alternating automaton  $\mathcal{A}$  runs on  $\Sigma$ -labeled  $D$ -trees. A *run* of  $\mathcal{A}$  over a  $\Sigma$ -labeled  $D$ -tree  $\langle T, \tau \rangle$  is a  $(T \times Q)$ -labeled  $\mathbb{N}$ -tree  $\langle T_r, r \rangle$ . Each node of  $T_r$  corresponds to a node of  $T$ . A node in  $T_r$ , labeled by  $(x, q)$ , describes a copy of the automaton that reads the node  $x$  of  $T$  and visits the state  $q$ . Note that many nodes of  $T_r$  can correspond to the same node of  $T$ . The labels of a node and its successors have to satisfy the transition function. Formally,  $\langle T_r, r \rangle$  satisfies the following: (1)  $\varepsilon \in T_r$  and  $r(\varepsilon) = \langle \varepsilon, q_{in} \rangle$ . (2) Let  $y \in T_r$  with  $r(y) = \langle x, q \rangle$  and  $\delta(q, \tau(x)) = \theta$ . Then there is a (possibly empty) set  $S = \{(c_0, q_0), (c_1, q_1), \dots, (c_{n-1}, q_{n-1})\} \subseteq D \times Q$ , such that  $S$  satisfies  $\theta$ , and for all  $0 \leq i \leq n-1$ , we have  $y \cdot i \in T_r$  and  $r(y \cdot i) = \langle x \cdot c_i, q_i \rangle$ .

A run  $\langle T_r, r \rangle$  is accepting if all its infinite paths satisfy the acceptance condition. Given a run  $\langle T_r, r \rangle$  and an infinite path  $\pi \subseteq T_r$ , let  $inf(\pi) \subseteq Q$  be such that  $q \in inf(\pi)$  if and only if there are infinitely many  $y \in \pi$  for which  $r(y) \in T \times \{q\}$ . That is,  $inf(\pi)$  contains exactly all the states that appear infinitely often in  $\pi$ . In Büchi and co-Büchi automata, the acceptance condition is  $\alpha \subseteq Q$ . A path  $\pi$  satisfies a Büchi condition  $\alpha$  iff  $inf(\pi) \cap \alpha \neq \emptyset$ , and satisfies a co-Büchi condition  $\alpha$  iff  $inf(\pi) \cap \alpha = \emptyset$ . In parity automata, the acceptance condition  $\alpha : Q \rightarrow \{1, \dots, k\}$  maps each vertex to a *color*. A path  $\pi$  satisfies a parity condition  $\alpha$  iff the minimal color that is visited infinitely often in  $\pi$  is even. Formally,  $\min\{i : inf(\pi) \cap \alpha^{-1}(i) \neq \emptyset\}$  is even. An automaton accepts a tree iff there exists a run that accepts it. We denote by  $L(\mathcal{A})$  the set of all  $\Sigma$ -labeled trees that  $\mathcal{A}$  accepts.

The alternating automaton  $\mathcal{A}$  is *nondeterministic* if for all the formulas that appear in  $\delta$ , if  $(c_1, q_1)$  and  $(c_2, q_2)$  are conjunctively related, then  $c_1 \neq c_2$ . (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of  $\{c\} \times Q$ , for each  $c \in D$ , in each disjunct). The automaton  $\mathcal{A}$  is *universal* if all the formulas that appear in  $\delta$  are conjunctions of atoms in  $D \times Q$ , and  $\mathcal{A}$  is *deterministic* if it is both nondeterministic and universal. The automaton  $\mathcal{A}$  is a *word automaton* if  $|D| = 1$ . Then, we can omit  $D$  from the specification of the automaton and denote the transition function of  $\mathcal{A}$  as  $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$ . If the word automaton is nondeterministic or universal, then  $\delta : Q \times \Sigma \rightarrow 2^Q$ , and we often extend  $\delta$  to sets of states and to finite words: for  $S \subseteq Q$ , we have that  $\delta(S, \varepsilon) = S$  and for a word  $w \in \Sigma^*$  and a letter  $\sigma \in \Sigma$ , we have  $\delta(S, w \cdot \sigma) = \delta(\delta(S, w), \sigma)$ . When  $\alpha \subseteq Q$ , we are sometimes interested in reachability via a nonempty path that visits  $\alpha$ . For this, we define  $\delta_\alpha : 2^Q \times \Sigma^+ \rightarrow 2^Q$  as follows. First,  $\delta_\alpha(S, \sigma) = \delta(S, \sigma) \cap \alpha$ . Then, for a word  $w \in \Sigma^+$ , we define  $\delta_\alpha(S, w \cdot \sigma) = \delta(\delta_\alpha(S, w), \sigma) \cup (\delta(S, w \cdot \sigma) \cap \alpha)$ . Thus, either  $\alpha$  is visited in the prefix of the run that reads  $w$  after leaving  $S$ , or the last state of the run is in  $\alpha$ . It is not hard to prove by an induction on the length of  $w$  that for all states  $q \in Q$ , we have that  $q \in \delta_\alpha(S, w)$  iff there is a run from  $S$  on  $w$  that reaches  $q$  and visits  $\alpha$  after leaving  $S$ . We sometimes refer also to word automata on finite words. There,  $\alpha \subseteq Q$  and a (finite) run is accepting if its last state is in  $\alpha$ .

We denote each of the different types of automata by three-letter acronyms in  $\{D, N, U, A\} \times \{F, B, C, P\} \times \{W, T\}$ , where the first letter describes the branching mode of the automaton (deterministic, nondeterministic, universal, or alternating), the second letter describes the acceptance condition (finite, Büchi, co-Büchi, or parity), and the third letter describes the object over which the automaton runs (words or trees). For example, UCT stands for a universal co-Büchi tree automaton.

### 3 Perspective Games with Notifications

Consider a game graph  $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$ . An *information satellite* for  $G$  (*satellite*, for short) is finite-state machine  $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2 \rangle$ , where  $O$  and  $I$  are *observation* and *information* alphabets,  $S$  is a finite set of states,  $s_0 \in S$  is an initial state,  $M : S \times O \rightarrow S$  is a deterministic transition function, and  $i_1, i_2 : S \rightarrow I \cup \{\varepsilon\}$  are information functions for Players 1 and 2, respectively, where  $\varepsilon \notin I$  is a special letter, standing for “no information”. We distinguish between *structural* satellites, where  $O = V$ , and *behavioral* satellites, where  $O = 2^{AP}$ . Intuitively, the satellite is executed during the play, updating its state according to the current vertex or its label, possibly notifying the players with information in  $I$ .

► **Example 1.** Assume there is an atomic proposition  $alarm \in AP$ . Both players can hear whenever an alarm is activated, but they do not know for how many rounds it is on. A satellite that informs the players about the activation of the alarm is  $\mathcal{I} = \langle 2^{\{alarm\}}, \{activated\}, S, s_0, M, i_1, i_2 \rangle$ , with  $S = \{s_0, s_1, s_2\}$ ,  $M(s_i, \neg alarm) = s_0$ , for all  $i \in \{0, 1, 2\}$ ,  $M(s_0, alarm) = s_1$ , and  $M(s_1, alarm) = M(s_2, alarm) = s_2$ . Thus, the satellite moves to  $s_1$  whenever a  $\neg alarm \cdot alarm$  pattern is read, and then moves to and stays in  $s_2$  as long as the alarm is on. When the alarm is deactivated, the satellite moves to  $s_0$ . Also,  $i_1(s_1) = i_2(s_1) = activated$ , and  $i_1(s_0) = i_1(s_2) = i_2(s_0) = i_2(s_2) = \varepsilon$ . Thus, when the satellite is in  $s_1$ , it notifies both players about the activation of the alarm.

A *perspective game with notifications* is a tuple  $\mathcal{G} = \langle G, \mathcal{I}, L \rangle$  where  $G$  and  $L$  are as in perspective games with no notifications, and  $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2 \rangle$  is a satellite. As in usual perspective games, PLAYER 1 aims for a play whose computation is in  $L$ , while PLAYER 2 aims for a play whose computation is in  $comp(L)$ . Now, however, the perspectives of the players contain, in addition to visits in their sets of vertices, also information from the satellite. Below we formalize this intuition.

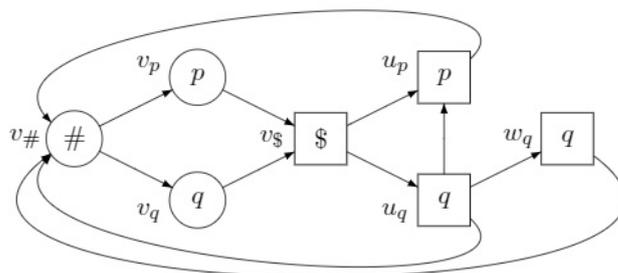
We define the function  $\zeta : V \rightarrow O$  that maps each vertex of  $G$  to the appropriate observation alphabet letter of  $\mathcal{I}$ . Thus, for every  $v \in V$ , we have that  $\zeta(v) = v$  if  $\mathcal{I}$  is structural, and  $\zeta(v) = \tau(v)$  if  $\mathcal{I}$  is behavioral. An *attributed path* in  $G$  is a sequence  $\eta \in (V \times S)^*$  obtained by attributing a path  $\rho = v_0, v_1, v_2, \dots, v_n \in V^*$  in  $G$  by the state in  $S$  that  $\mathcal{I}$  visits when a play proceeds along  $\rho$ . Formally,  $\eta = \langle v_0, s_0 \rangle, \langle v_1, s_1 \rangle, \dots, \langle v_n, s_n \rangle$  is such that for all  $1 \leq i \leq n$ , we have that  $s_i = M(s_{i-1}, \zeta(v_i))$ . Note that first the play proceeds from  $v_{i-1}$  to  $v_i$ , and then the satellite reads  $\zeta(v_i)$  and proceeds accordingly. We use  $Last(\eta)$  to refer to  $v_n$ . Let  $Prefs^I(G) \subseteq (V \times S)^*$  be the set of nonempty attributed prefixes of plays in  $G$ . For  $j \in \{1, 2\}$ , let  $Prefs_j^I(G) = \{\eta \in Prefs^I(G) : Last(\eta) \in V_j\}$ . For a prefix  $\eta \in Prefs^I(G)$ , the *rich perspective of PLAYER  $j$  on  $\eta$* , denoted  $Persp_j^I(\eta)$ , is the restriction of  $\eta$  to vertices in  $V_j$  and notifications of  $\mathcal{I}$  that occur in vertices not in  $V_j$ . Formally, the function  $info_j : (V \times S) \rightarrow V_j \cup I$  describes the information added to PLAYER  $j$  in each round. For all  $\langle v, s \rangle \in V \times S$ , if  $v \in V_j$ , then  $info_j(\langle v, s \rangle) = v$ ; if  $v \notin V_j$ , then  $info_j(\langle v, s \rangle) = i_j(s)$ . Note that in the latter case, it may be that  $i_j(s) = \varepsilon$ . Thus, if  $\eta = \langle v_0, s_0 \rangle, \langle v_1, s_1 \rangle, \dots, \langle v_n, s_n \rangle$ , then  $Persp_j^I(\eta) = info_j(\langle v_0, s_0 \rangle) \cdot info_j(\langle v_1, s_1 \rangle) \cdots info_j(\langle v_n, s_n \rangle)$ . Note that  $\varepsilon$  does not contribute

letters to  $\text{Persp}_j^I(\eta)$ , and so the length of  $\text{Persp}_j^I(\eta)$  is the number of the vertices in  $V_j$  in  $\eta$  plus the number of vertices not in  $V_j$  in which the satellite provides to PLAYER  $j$  information in  $I$ .

► **Example 2.** Consider the alarm activation satellite described in Example 1, and consider a game graph  $G$ . Let  $v_2^\uparrow$  and  $v_2^\downarrow$  be vertices of PLAYER 2 with  $\text{alarm} \in \tau(v_2^\uparrow)$  and  $\text{alarm} \notin \tau(v_2^\downarrow)$ . Then, the rich perspective of PLAYER 1 on the path  $v_2^\downarrow, v_2^\downarrow, v_2^\downarrow, v_2^\downarrow, v_2^\uparrow, v_2^\uparrow, v_2^\uparrow, v_2^\uparrow, v_2^\uparrow, v_2^\uparrow$  is  $\bullet, \bullet$ , reflecting the two activations of the alarm during its traversal. Now, if  $v_1^\uparrow \in V_1$ , and  $\text{alarm} \in \tau(v_1^\uparrow)$ , then the rich perspective of PLAYER 1 on  $v_2^\downarrow, v_2^\downarrow, v_1^\uparrow, v_2^\downarrow, v_2^\uparrow, v_2^\uparrow, v_1^\uparrow, v_2^\downarrow, v_1^\uparrow, v_2^\uparrow, v_2^\uparrow, v_2^\downarrow, v_2^\uparrow$  is  $v_1^\uparrow, \bullet, v_1^\uparrow, v_1^\uparrow$ .

We denote the perspective of PLAYER  $j$  on prefixes in  $\text{Prefs}_j^I(G)$  by  $\text{PPrefs}_j^I(G)$ ; thus  $\text{PPrefs}_j^I(G) = \{\text{Persp}_j^I(\eta) : \eta \in \text{Prefs}_j^I(G)\}$ . A *perspective strategy* for PLAYER  $j$  (P-strategy for short) is then a function  $f_j : \text{PPrefs}_j^I(G) \rightarrow V$  such that for all  $\rho \in \text{PPrefs}_j^I(G)$ , we have that  $\langle \text{Last}(\rho), f_j(\rho) \rangle \in E$ . That is, a perspective strategy for PLAYER  $j$  maps her perspective prefixes of plays that end in a vertex  $v \in V_j$  to a successor of  $v$ . The definitions of the outcome of F or P-strategies and F or P-winning are similar to the definitions in perspective games with no notifications, with  $\text{Persp}_j^I$  instead of  $\text{Persp}_j$ .

► **Example 3.** Consider the game graph  $G$  appearing in Figure 1. For simplicity, we assume that the atomic propositions in  $AP$  are mutually exclusive, and thus each vertex is labeled by a letter in  $\Sigma = \{p, q, \#, \$\}$ .



■ **Figure 1** The game graph  $G$  over  $\{p, q, \#, \$\}$ . The vertices of PLAYER 1 are circles, and those of PLAYER 2 are squares. The initial vertex is  $v_\#$ .

Note that whenever the token reaches  $v_\$$ , there are four possible sub-computations it may generate before returning to  $v_\#$ ; these are  $\$ \cdot p \cdot \#$ ,  $\$ \cdot q \cdot \#$ ,  $\$ \cdot q \cdot p \cdot \#$  and  $\$ \cdot q \cdot q \cdot \#$ . Let  $\mathcal{G}_1 = \langle G, \varphi_1 \rangle$  be a perspective game with  $\varphi_1 = G(((q \wedge Xq) \rightarrow XXXq) \wedge ((q \wedge Xp) \rightarrow XXXp))$ . That is,  $\varphi_1$  requires every  $q \cdot q$  subword to be followed by a subword in  $\Sigma \cdot q$ , and every  $q \cdot p$  subword to be followed by  $\Sigma \cdot p$ . It is easy to see that PLAYER 1 cannot (P, F)-win  $\mathcal{G}_1$ , because she is unable to distinguish between the different possible sub-computations, and thus every P-strategy of hers chooses the same successor of  $v_\#$  for all four cases. Now consider the perspective game with notifications  $\mathcal{G}'_1 = \langle G, \mathcal{I}_1, \varphi_1 \rangle$  where  $\mathcal{I}_1$  is a structural satellite that notifies PLAYER 1 whenever a visit in  $w_q$  occurs. The information from the satellite restricts the possibilities; when PLAYER 1 gets a notification, she knows that the last sub-computation is  $\$ \cdot q \cdot q \cdot \#$ . When she does not get a notification, she knows that the last sub-computation is one of the other possibilities. Therefore, PLAYER 1 (P, F)-wins  $\mathcal{G}'_1$ , as she can distinguish between the sub-computations  $\$ \cdot q \cdot q \cdot \#$  and  $\$ \cdot q \cdot p \cdot \#$ , and can choose the successor of  $v_\#$  after each visit in it in a way that satisfies  $\varphi_1$ .

Let  $\mathcal{G}_2 = \langle G, \varphi_2 \rangle$  be a perspective game with  $\varphi_2 = G((\$ \wedge Xp) \rightarrow XXXp) \wedge ((q \wedge Xp) \rightarrow XXXq)$ . Again, PLAYER 1 cannot  $(P, F)$ -win  $\mathcal{G}_2$ . Now consider the perspective game with notifications  $\mathcal{G}'_2 = \langle G, \mathcal{I}_2, \varphi_2 \rangle$ , where  $\mathcal{I}_2$  is a behavioral satellite that notifies PLAYER 1 whenever the computation generated so far is a word in the regular language  $(p + q + \# + \$)^* \cdot \$ \cdot p$ . Now, when PLAYER 1 gets a notification, she knows that the last sub-computation is  $\$ \cdot p \cdot \#$ , and when she does not get a notification, she knows that the last sub-computation is one of the other possibilities. Therefore, PLAYER 1  $(P, F)$ -wins  $\mathcal{G}'_2$ . Indeed, PLAYER 1 can distinguish between the sub-computations  $\$ \cdot p \cdot \#$  and  $\$ \cdot q \cdot p \cdot \#$ , and can choose the successor of  $v_\#$  after each visit in it in a way that satisfies  $\varphi_2$ .

Note that PLAYER 1 cannot P-win the games  $\langle G, \mathcal{I}_1, \varphi_2 \rangle$  and  $\langle G, \mathcal{I}_2, \varphi_1 \rangle$ . Indeed,  $\mathcal{I}_1$  does not enable PLAYER 1 to distinguish between the sub-computations  $\$ \cdot p \cdot \#$  and  $\$ \cdot q \cdot p \cdot \#$ , and  $\mathcal{I}_2$  does not enable PLAYER 1 to distinguish between the sub-computations  $\$ \cdot q \cdot q \cdot \#$  and  $\$ \cdot q \cdot p \cdot \#$ . Therefore, in both games, a P-strategy of PLAYER 1 chooses the same successor of  $v_\#$  in these undistinguishable cases.

Example 3 shows that, as is the case in perspective games with no notifications [8], P-strategies with no notifications are weaker than P-strategies with notifications, which are weaker than F-strategies. It also shows that perspective games with notifications are not determined. That is, there are perspective games with notifications where both PLAYER 1 and PLAYER 2 do not have P-winning strategies. Also, the visibility type of PLAYER 2 does not matter. Essentially, it follows from the fact that if a perspective strategy of PLAYER 1 loses against an F-strategy  $f_2$  of PLAYER 2, then it also loses to a P-strategy of PLAYER 2 that is induced from  $f_2$ . The formal proofs of the above properties are similar to the case of perspective games with no notifications [8] and we leave them to the full version.

Since the visibility type of PLAYER 2 does not matter, we can omit it from our notation and talk about PLAYER 1 P-winning a game. Also, specifying satellites, we remove the function  $i_2$  from their description.

#### 4 Deciding Perspective Games with Notifications

Consider a game  $\mathcal{G} = \langle G, \mathcal{I}, L \rangle$ , for a game graph  $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$  and a satellite  $\mathcal{I} = \langle O, I, S, s_0, M, i_1 \rangle$ . For a regular expression  $R$  over the alphabet  $V$ , an  $R$ -path from  $v$  is a finite path  $v_1, \dots, v_k \in L(R)$  in  $G$  such that  $v_1 = v$ . For a subset  $X \subseteq V$ , an  $X^\omega$ -path from  $v$  is an infinite path  $v_1, v_2, \dots \in X^\omega$  in  $G$  with  $v_1 = v$ . Note, for example, that when PLAYER 1 moves the token to a vertex  $v \in V_2$ , the token may traverse a  $(V_2^+ \cdot V_1)$ -path  $\rho$  from  $v$ , in which case it returns to  $V_1$  in  $\text{Last}(\rho)$ , or it may traverse a  $V_2^\omega$ -path from  $v$ , in which case it never returns to a vertex in  $V_1$ . For a regular expression  $R$  over the alphabet  $V \times S$ , an  $R$ -path from  $\langle v, s \rangle$  is an attributed path  $\langle v_1, s_1 \rangle, \dots, \langle v_k, s_k \rangle \in L(R)$  in  $G$  with  $v_1 = v$  and  $s_1 = s$ . For such a path  $\rho$ , we denote its projections on  $V$  and  $S$  by  $\rho|_V$  and  $\rho|_S$ , respectively.

Consider the satellite  $\mathcal{I}$ . For  $\sigma \in I \cup \{\varepsilon\}$ , we denote by  $S_\sigma$  the set of states in  $\mathcal{I}$  in which PLAYER 1 is notified  $\sigma$ . That is,  $S_\sigma = \{s \in S : i_1(s) = \sigma\}$ . Then,  $S_I = \bigcup_{\sigma \in I} S_\sigma$  is the set of states in which PLAYER 1 is notified some information. Equivalently,  $S_I = S \setminus S_\varepsilon$ .

We focus on games in which the winning condition  $L$  is given by a UCW. For simplicity, we denote them by  $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U} \rangle$ , for a UCW  $\mathcal{U}$ . Let  $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha \rangle$ . In order for PLAYER 1 to P-win  $\mathcal{G}$ , her objective in the beginning of the game is to force a token that is placed in  $v_0$  into computations that  $\mathcal{U}$  accepts from  $q_0$  with the satellite being in state  $s_0$ . We can describe this objective by the triple  $\langle v_0, q_0, s_0 \rangle$ . As the play progresses, the objective of PLAYER 1 is updated. Moreover, as  $\mathcal{U}$  is universal, the objective may contain several such triples. Below we formalize this intuition.

Consider a UCW  $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha \rangle$ , a state  $q \in Q$ , and a state  $s \in S$ . Suppose that the token is placed in some vertex  $v \in V_1$ , the objective of PLAYER 1 is to force the token into computations in  $L(\mathcal{U}^q)$ , and the satellite is in state  $s$  after seeing  $\zeta(v)$ . Assume further that PLAYER 1 chooses to move the token to a successor  $v'$  of  $v$  and that  $s' = M(s, \zeta(v'))$ . We distinguish between two cases.

1.  $v' \in V_1$ . Then, the new objective of PLAYER 1 is to force the token in  $v'$  into computations in  $L(\mathcal{U}^{q'})$ , for all states  $q' \in \delta(q, \tau(v))$ , with the satellite being in state  $s'$ .
2.  $v' \in V_2$ . Then, there are three cases:
  - a. There is a  $V_2^\omega$ -path  $\rho$  from  $v'$  with  $\tau(\rho) \notin L(\mathcal{U}^{q'})$  for some  $q' \in \delta(q, \tau(v))$ . We then say that  $v'$  is a trap for  $\langle v, q \rangle$ . Indeed, PLAYER 2 can stay in vertices in  $V_2$  and force the token into a computation not in  $L(\mathcal{U}^{q'})$ . Note that once PLAYER 1 chooses a vertex that is a trap for  $\langle v, q \rangle$ , PLAYER 2 has a strategy to win the game.
  - b.  $v'$  is not a trap for  $\langle v, q \rangle$ , yet there is no  $(V_2^+ \cdot V_1)$ -path from  $v'$ . That is, all paths from  $v'$  stay in vertices in  $V_2$  and are in  $L(\mathcal{U}^{q'})$  for all  $q' \in \delta(q, \tau(v))$ . We then say that  $v'$  is safe for  $\langle v, q \rangle$ . Indeed, PLAYER 2 stays in vertices in  $V_2$  and all the possible plays induce a computation in  $L(\mathcal{U}^q)$ . Note that once PLAYER 1 chooses a safe vertex for  $\langle v, q \rangle$ , her objective is fulfilled regardless of the strategy of PLAYER 2.
  - c.  $v'$  is neither a trap nor safe for  $\langle v, q \rangle$ , in which case:
    - i. For every  $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$  PLAYER 1 should force a token that is placed in  $v''$  into computations in  $L(\mathcal{U}^{q'})$ , for all states  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ , with the satellite being in state  $s''$ . Note that for all  $\langle \hat{v}, \hat{s} \rangle$  along  $\rho$ , we have  $\text{info}_1(\langle \hat{v}, \hat{s} \rangle) = \varepsilon$ , and so the visit in  $v''$  is the first event that PLAYER 1 observes after placing the token in  $v'$ .
    - ii. For every  $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$ , PLAYER 1 should force a token that is placed in  $v''$  with the satellite being in state  $s''$  into computations in  $L(\mathcal{U}^{q'})$ , for all states  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ . Note that for all  $\langle \hat{v}, \hat{s} \rangle$  along  $\rho$ , we have  $\text{info}_1(\langle \hat{v}, \hat{s} \rangle) = \varepsilon$ , and so  $i_1(s'')$  is the first event that PLAYER 1 observes after placing the token in  $v'$ . Also note that  $\rho$  might be empty, in particular when PLAYER 1 moves the token to a vertex in  $V_2$  that invokes a notification of  $\mathcal{I}$ . In this case,  $\langle v', s' \rangle = \langle v'', s'' \rangle$ .

The above analysis induces the definition of *updated objectives*: Consider a triple  $\langle v, q, s \rangle \in V_1 \times Q \times S$ , standing for an objective of PLAYER 1 to force a token placed on  $v$  to be accepted by  $\mathcal{U}^q$  with the satellite being in state  $s$ . For a successor  $v'$  of  $v$ , we define the set  $S_{v,q,s}^{v'} \subseteq (V \times Q \times S \times \{\perp, \top\}) \cup \{\mathbf{false}\}$  of objectives that PLAYER 1 has to satisfy in order to fulfil her  $\langle v, q, s \rangle$  objective after choosing to move the token to  $v'$ . Also, for a triple  $\langle v, q, s \rangle \in V_2 \times Q \times S$ , we define the set  $S_{v,q,s} \subseteq V \times Q \times S \times \{\perp, \top\}$  of objectives that PLAYER 1 has to satisfy in order to fulfil her  $\langle v, q, s \rangle$  objective for every successor that PLAYER 2 might choose for  $v$ . In both cases, the  $\{\perp, \top\}$  flag in the objectives is used for tracking visits in  $\alpha$ : an updated objective  $\langle v'', q', s'', c \rangle \in S_{v,q,s}^{v'}$  has  $c = \top$  if PLAYER 2 can force a visit in  $\alpha$  when  $\mathcal{U}$  runs from  $q$  to  $q'$  along a word that labels a path from  $v$  via  $v'$  to  $v''$ .

Formally, for a triple  $\langle v, q, s \rangle \in V \times Q \times S$  we define the set of updated objectives as follows. Let  $s' = M(s, \zeta(v'))$ .

1. If  $v \in V_1$  and  $E(v, v')$ , we distinguish between three cases.
  - a. If  $v'$  is a trap for  $\langle v, q \rangle$ , then  $S_{v,q,s}^{v'} = \{\mathbf{false}\}$ .
  - b. If  $v'$  is safe for  $\langle v, q \rangle$ , then  $S_{v,q,s}^{v'} = \emptyset$ .
  - c. Otherwise, a tuple  $\langle v'', q', s'', c \rangle$  is in  $S_{v,q,s}^{v'}$  iff one of the following holds.
    - i.  $v' \in V_1$ ,  $v'' = v'$ ,  $q' \in \delta(q, \tau(v))$ , and  $s'' = s'$ . Then,  $c = \top$  iff  $q' \in \alpha$ .

- ii.  $v' \in V_2$ , and there is an  $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$  such that  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ . Then,  $c = \top$  iff there is an  $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$  such that  $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$ .
  - iii.  $v' \in V_2$ , and there is an  $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$  such that  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ . Then,  $c = \top$  iff there is an  $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v', s' \rangle$  such that  $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$ .
2. If  $v \in V_2$ , a tuple  $\langle v'', q', s'', c \rangle$  is in  $S_{v,q,s}$  iff one of the following holds.
- a. There is an  $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v, s \rangle$  such that  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ . Then,  $c = \top$  iff there is an  $(V_2 \times S_\varepsilon)^+ \cdot (V_1 \times S)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v, s \rangle$  such that  $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$ .
  - b. There is an  $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v, s \rangle$  such that  $q' \in \delta(q, \tau(v \cdot \rho|_V))$ . Then,  $c = \top$  iff there is an  $(V_2 \times S_\varepsilon)^* \cdot (V_2 \times S_I)$ -path  $\rho \cdot \langle v'', s'' \rangle$  from  $\langle v, s \rangle$  such that  $q' \in \delta_\alpha(q, \tau(v \cdot \rho|_V))$ .

The notion of updated objectives is the key to our algorithm for deciding P-winning in perspective games with notifications. Recall that a perspective strategy for PLAYER 1 is a function  $f_1 : \text{PPrefs}_1(G) \rightarrow V$  such that for all  $\rho \in \text{PPrefs}_1(G)$ , we have that  $\langle \text{Last}(\rho), f_1(\rho) \rangle \in E$ , where  $\text{PPrefs}_1(G)$  contains words in  $V_1 \cup I$  that end with a vertex in  $V_1$ . Accordingly, we describe a strategy for PLAYER 1 by a  $(V \cup \{\odot\})$ -labeled  $(V_1 \cup I)$ -tree, where the letter  $\odot$  label nodes  $x \notin \text{PPrefs}_1(G)$ , namely nodes  $x \in (V_1 \cup I)^* \cdot I$ . Formally, a  $(V \cup \{\odot\})$ -labeled  $(V_1 \cup I)$ -tree  $\langle (V_1 \cup I)^*, \eta \rangle$  is a P-strategy of PLAYER 1 if for all  $\rho \in (V_1 \cup I)^*$  and  $v \in V_1$ , we have that  $\eta(\rho \cdot v) = v'$ , where  $v' \in V$  is such that  $E(v, v')$ , and for all  $\sigma \in I$  we have that  $\eta(\rho \cdot \sigma) = \odot$ , indicating PLAYER 1 does not move the token when she receives the  $\sigma$  notification, and just keeps this notification in mind.

► **Theorem 4.** *Let  $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U} \rangle$  be a game with notifications, where  $G$  is a game graph,  $\mathcal{I} = \langle O, I, S, s_0, M, i_1 \rangle$  is a satellite, and  $\mathcal{U}$  is a UCW. We can construct a UCT  $\mathcal{A}_{\mathcal{G}}$  over  $(V \cup \{\odot\})$ -labeled  $(V_1 \cup I)$ -trees such that  $\mathcal{A}_{\mathcal{G}}$  accepts a  $(V \cup \{\odot\})$ -labeled  $(V_1 \cup I)$ -tree  $\langle (V_1 \cup I)^*, \eta \rangle$  iff  $\langle (V_1 \cup I)^*, \eta \rangle$  is a winning P-strategy for PLAYER 1. The size of  $\mathcal{A}_{\mathcal{G}}$  is polynomial in  $|G|$ ,  $|\mathcal{I}|$ , and  $|\mathcal{U}|$ .*

**Proof.** Let  $\mathcal{U} = \langle 2^{AP}, Q, q_0, \delta, \alpha \rangle$ . We define  $\mathcal{A}_{\mathcal{G}} = \langle V \cup \{\odot\}, V_1 \cup I, Q', q'_0, \delta', \alpha' \rangle$ , where:

1.  $Q' = V \times Q \times S \times \{\perp, \top\}$ . Intuitively, when  $\mathcal{A}_{\mathcal{G}}$  is in state  $\langle v, q, s, c \rangle$  it accepts strategies that force a token placed on  $v$  into a computation accepted by  $\mathcal{U}^q$  with the satellite being in state  $s$ . The flag  $c$  is used for tracking visits in  $\alpha$ .
2.  $q'_0 = \langle v_0, q_0, s_0, \perp \rangle$ .
3. The transitions are defined, for all states  $\langle v, q, s, c \rangle \in V_1 \times Q \times S \times \{\perp, \top\}$ , as follows.
  - a. If  $v \in V_1$ , then  $\delta'(\langle v, q, s, c \rangle, \odot) = \text{false}$ , and for every  $v' \in V$  we have the following transitions.
    - i. If  $S_{v,q,s}^{v'} = \{\text{false}\}$  or  $\neg E(v, v')$ , then  $\delta'(\langle v, q, s, c \rangle, v') = \text{false}$ .
    - ii. If  $S_{v,q,s}^{v'} = \emptyset$ , then  $\delta'(\langle v, q, s, c \rangle, v') = \text{true}$ .
    - iii. Otherwise,  $\delta'(\langle v, q, s, c \rangle, v') =$

$$\bigwedge_{\langle v'', q', s'', c' \rangle \in S_{v,q,s}^{v'} : v'' \in V_1} (v'', \langle v'', q', s'', c' \rangle) \wedge \bigwedge_{\langle v'', q', s'', c' \rangle \in S_{v,q,s}^{v'} : v'' \in V_2} (i_1(s''), \langle v'', q', s'', c' \rangle).$$

- b. If  $v \in V_2$ , then for all  $v' \in V$ , we have that  $\delta'(\langle v, q, s, c \rangle, v') = \text{false}$ .

Also,  $\delta'(\langle v, q, s, c \rangle, \odot) =$

$$\bigwedge_{\langle v'', q', s'', c' \rangle \in S_{v,q,s} : v'' \in V_1} (v'', \langle v'', q', s'', c' \rangle) \wedge \bigwedge_{\langle v'', q', s'', c' \rangle \in S_{v,q,s} : v'' \in V_2} (i_1(s''), \langle v'', q', s'', c' \rangle).$$

Thus, for every updated objective  $\langle v'', q', s'', c' \rangle$ , the automaton  $\mathcal{A}_{\mathcal{G}}$  sends a copy in state  $\langle v'', q', s'', c' \rangle$  to direction  $v''$  if  $v'' \in V_1$ , and to direction  $i_1(s'')$ , if  $v'' \in V_2$ . Note that several updated requirements may be sent to the same direction. In particular, in addition to multiple copies sent to the same direction due to universal branches in  $\mathcal{U}$ , a direction  $\sigma \in I$  may “host” updated objectives associated with different vertices in  $V_2$ . Intuitively, such vertices are indistinguishable by PLAYER 1.

4.  $\alpha' = V \times Q \times S \times \{\top\}$ . Recall that a  $\top$  flag indicates that PLAYER 2 may reach the  $Q$ -element in an updated objective traversing a path that visits  $\alpha$ . Accordingly, the co-Büchi requirement to visit  $\alpha$  only finitely many times amounts to a requirement to visit states with  $\top$  only finitely many times. ◀

Theorem 4 gives us an upper bound on the problem of deciding whether PLAYER 1 P-wins a perspective game with notifications.

► **Theorem 5.** *Deciding whether PLAYER 1 P-wins a perspective game with notifications  $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U} \rangle$ , for a UCW  $\mathcal{U}$ , is EXPTIME-complete, and can be solved in time polynomial in  $|G|$  and  $|\mathcal{I}|$ , and exponential in  $|\mathcal{U}|$ .*

**Proof.** Let  $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U} \rangle$  and  $\mathcal{I} = \langle O, I, S, s_0, M, i_1 \rangle$ . By Theorem 4, we can construct a UCT  $\mathcal{A}_{\mathcal{G}}$  over  $(V \cup \{\circlearrowleft\})$ -labeled  $(V_1 \cup I)$ -trees such that  $L(\mathcal{A}_{\mathcal{G}})$  is not empty iff there is a winning P-strategy for PLAYER 1 in  $\mathcal{G}$ . The size of  $\mathcal{A}_{\mathcal{G}}$  is polynomial in  $|G|$ ,  $|\mathcal{I}|$  and  $|\mathcal{U}|$ .

We construct an NBT  $\mathcal{A}'_{\mathcal{G}}$  over  $(V \cup \{\circlearrowleft\})$ -labeled  $(V_1 \cup I)$ -trees such that  $L(\mathcal{A}'_{\mathcal{G}})$  is not empty iff there is a winning P-strategy for PLAYER 1 in  $\mathcal{G}$ . The size of  $\mathcal{A}'_{\mathcal{G}}$  is polynomial in  $|G|$  and  $|\mathcal{I}|$ , and is exponential in  $|\mathcal{U}|$ . As has been the case in the setting with no notifications [8], the transformation from  $\mathcal{A}_{\mathcal{G}}$  to  $\mathcal{A}'_{\mathcal{G}}$  uses the fact that  $\mathcal{A}_{\mathcal{G}}$  is deterministic in the  $V$  and  $S$  components, in order to generate, following the construction of [10], an NBT that it is polynomial in  $|G|$  and  $|\mathcal{I}|$  and exponential only in  $|\mathcal{U}|$ . Since the nonemptiness problem for an NBT can be solved in quadratic time, the specified complexity follows.

Since perspective games with notifications are a special case of perspective game (technically, with a satellite that only outputs  $\varepsilon$ ), EXPTIME-hardness of the former implies an EXPTIME lower bound for our setting. ◀

Since an LTL  $\psi$  formula can be translated to a UCW  $\mathcal{U}_{\psi}$  with an exponential blow up (for example, by translating  $\neg\psi$  to an NBW [17], and then dualizing the NBW), Theorem 5 implies a 2EXPTIME upper bound for perspective games with notifications in which the winning condition is given by an LTL formula. Also, as has been the case in [8], it is possible to refine the  $\{\perp, \top\}$  flag in the updated objectives to maintain the minimal parity color that is visited, and adjust the construction to games in which the winning condition is given by a UPW. The complexity stays exponential in the automaton. Formally, we have the following.

► **Theorem 6.** *Deciding whether PLAYER 1 P-wins a perspective game with notifications  $\mathcal{G} = \langle G, \mathcal{I}, \mathcal{U} \rangle$ , for a UPW  $\mathcal{U}$ , is EXPTIME-complete, and can be solved in time polynomial in  $|G|$  and  $|\mathcal{I}|$ , and exponential in  $|\mathcal{U}|$ .*

**Proof.** The updated objectives defined for the case where the winning condition is given by a UCW contain a flag that records visits in the co-Büchi condition. When  $\mathcal{U}$  is a UPW with  $k$  colors, we define the flag such that it records the minimal color visited instead. That is,  $S_{v,q,s}^{v'}, S_{v,q,s} \subseteq (V \times Q \times S \times \{1, \dots, k\}) \cup \{\mathbf{false}\}$ , is such that for every updated objective  $\langle v'', q', s'', c \rangle \in S_{v,q,s}^{v'} \cup S_{v,q,s}$ , PLAYER 2 can force a path from  $v$  (via  $v'$ ) to  $v''$  in which the

minimal color visited in the run of  $\mathcal{U}$  along it from  $q$  to  $q'$  is  $c$ . We then use a construction that is similar to the one in the proof of Theorem 4 to construct a UPT  $\mathcal{A}_{\mathcal{G}}$  over  $(V \cup \{\circ\})$ -labeled  $(V_1 \cup I)$ -trees such that  $L(\mathcal{A}_{\mathcal{G}})$  is not empty iff there is a winning P-strategy for PLAYER 1 in  $\mathcal{G}$ . The size of  $\mathcal{A}_{\mathcal{G}}$  is polynomial in  $|G|$ ,  $|I|$  and  $|\mathcal{U}|$ .

By [10], APT emptiness can be reduced to UCT emptiness with a polynomial blow up. From there, determinism in the  $V$ -component implies the required complexity.  $\blacktriangleleft$

## 5 Examples of Information Satellites

Consider a game graph  $G = \langle AP, V_1, V_2, v_0, E, \tau \rangle$ . Recall that a *structural satellite* for  $G$  is a satellite  $\mathcal{I} = \langle O, I, S, s_0, M, i_1 \rangle$  with  $O = V$ . Thus, the satellite can view the state in which the play is, and can decide about outputs to PLAYER 1 based on this visibility. Then, a *behavioral satellite* for  $G$  has  $O = 2^{AP}$ . Thus, the satellite can only observe the labels of vertices, and its outputs to PLAYER 1 are based only on these labels. In this section we describe some natural structural and behavioral satellites.

### 5.1 Structural Information Satellites

**A visible subset of vertices.** As discussed in Section 1, in some settings there is a subset of vertices  $I_1 \subseteq V_2$  such that PLAYER 1 is notified whenever the play visits a vertex in  $I_1$ . Then, the satellite is  $\langle V, I_1, V, v_0, M, i_1 \rangle$ , where for all  $v, u \in V$ , we have that  $M(v, u) = u$ ,  $i_1(v) = v$  if  $v \in I_1$ , and  $i_1(v) = \varepsilon$ , otherwise. Thus, the state of the satellite follows the vertex of the game, and it produces an output during visits in  $I_1$ . Note that PLAYER 1 is notified not only about visits in  $I_1$ , but also about the specific vertex that is visited. Alternatively, we could define the satellite with output *in* only,  $i_1(v) = \text{in}$  if  $v \in I_1$ , and  $i_1(v) = \varepsilon$ , otherwise. Here, PLAYER 1 is notified that some vertex in  $I_1$  has been visited, with no information about which vertex it is.

**Observation-based uncertainty.** Assume that there is a subset of the atomic propositions  $AP_1 \subseteq AP$ , such that PLAYER 1 observes the assignments to  $AP_1$  in PLAYER 2's vertices. A corresponding satellite is  $\langle V, 2^{AP_1}, V, v_0, M, i_1 \rangle$ , where for all  $v, u \in V$ , we have that  $M(v, u) = u$ ,  $i_1(v) = \tau(v) \cap AP_1$  if  $v \in V_2$ , and  $i_1(v) = \varepsilon$ , otherwise. Note that this case combines the transverse visibility of perspective games with the longitudinal visibility in observation-based games. Indeed, when the token is in PLAYER 2's vertices, PLAYER 1's visibility is observation based. In particular, PLAYER 1 knows the number of vertices visited, yet cannot distinguish between paths that differ only in assignments to atomic propositions in  $AP \setminus AP_1$ . It is not hard to see that when  $AP_1 = AP$ , then, as the winning condition is behavioral (that is, refers to  $AP$  rather than to  $V$ ), the setting coincides with games with full visibility. Also, note that even though the notifications of the satellite are in  $2^{AP_1}$ , we could not define it as a behavioral information satellite.

**Visible switches among regions.** Assume that the vertices in  $V_2$  is partitioned into disjoint *regions*  $V_2^1, \dots, V_2^k$ . For example, the regions may correspond to modules or procedures. If PLAYER 1 is notified upon entry to the different regions, then the corresponding satellite is  $\langle V, \{1, \dots, k\}, S, \langle v_0, \circ \rangle, M, i_1 \rangle$ , where  $S = (V_1 \times \{\circ\}) \cup (V_2 \times \{\circ, \bullet\})$ . Thus, the state space of the satellite has one copy of the vertices in  $V_1$  and two copies of the vertices in PLAYER 2. Then,  $M$  and  $i_1$  are as follows. For a vertex  $v \in V_2$ , let  $\text{reg}(v)$  be the region of  $v$ ; thus  $v \in V_2^{\text{reg}(v)}$ . Then, for all  $v, u \in V$  and  $j \in \{\circ, \bullet\}$ , we have that  $M(\langle v, j \rangle, u) = \langle u, \circ \rangle$  if  $u \in V_1$  or  $\text{reg}(v) = \text{reg}(u)$ , and  $M(\langle v, j \rangle, u) = \langle u, \bullet \rangle$  if  $\text{reg}(v) \neq \text{reg}(u)$ . Also, for every  $\langle v, j \rangle \in S$  we

have that  $i_1(\langle v, j \rangle) = \text{reg}(v)$  if  $j = \bullet$ , and  $i_1(\langle v, j \rangle) = \varepsilon$ , otherwise. As in the case of a visible subset of vertices, the satellite can notify PLAYER 1 only about a switch in a region, without specifying which region it is. Then, the satellite has only output  $\bullet$ , and  $i_1(\langle v, j \rangle) = \bullet$  if  $j = \bullet$ , and  $i_1(\langle v, j \rangle) = \varepsilon$ , otherwise. Note that in both case, PLAYER 1 is not notified about the number of rounds that PLAYER 2 is spending in each region, and only about switches among them.

An interesting variant of the above is a satellite that notifies PLAYER 1 whenever PLAYER 2 loops in a vertex. Note that this is a special case of the above, where each vertex of  $V_2$  has its own region, with a dual  $\{\circ, \bullet\}$  notification. Namely, we let PLAYER 1 know when there is no change in the region. Then, the satellite is  $\langle V, \{\bullet\}, S, \langle v_0, \circ \rangle, M, i_1 \rangle$ , where  $i_1$  is as above, yet for every  $v, u \in V$  and  $j \in \{\circ, \bullet\}$ , we have that  $M(\langle v, j \rangle, u) = \langle u, \circ \rangle$  if  $u \in V_1$  or  $v \neq u$ , and  $M(\langle v, j \rangle, u) = \langle u, \bullet \rangle$ , otherwise.

## 5.2 Behavioral Information Satellites

**Visible regular properties.** Assume there is a property, given by a regular language  $R$  over  $2^{AP}$ , such that PLAYER 1 is notified whenever the computation generated since the beginning of the play is in  $R$ . For example, if  $AP = \{p, q\}$ , the property may be  $\mathbf{true}^* \cdot p \cdot (\neg q)^*$ , thus we want to notify PLAYER 1 whenever a vertex satisfying  $p$  has been visited with no visit in a vertex satisfying  $q$  following this visit. Then, if  $A_R = \langle 2^{AP}, S, s_0, M, F \rangle$  is a DFW that recognizes  $R$ , an appropriate satellite is  $\mathcal{I} = \langle 2^{AP}, \{\bullet\}, S, M(s_0\tau(v_0)), M, i_1 \rangle$ , where for every  $s \in S$ , we have that  $i_1(s) = \bullet$  if  $s \in F$ , and  $i_1(s) = \varepsilon$ , otherwise. Note that the initial state of the satellite is the state of  $A_R$  after reading the label of  $v_0$ . Indeed, notifications inform PLAYER 1 about the membership of the computation up to (and including) the vertex where the token visits. A useful special case of regular properties are these of the form  $\mathbf{true}^* \cdot R$ , for a regular language  $R$  over  $2^{AP}$ . Thus, PLAYER 1 is notified whenever the computation generated since the beginning of the play has a suffix in  $R$ . As we discuss in Section 6, handling of the two types of notifications is of different complexity.

The above can be generalized to multiple regular languages  $R_1, \dots, R_k$  over  $2^{AP}$ , where for every  $1 \leq i \leq k$ , PLAYER 1 is notified whenever the computation generated since the beginning of the play is in  $R_i$ . Indeed, if for every  $1 \leq i \leq k$ , the DFW  $A_i = \langle 2^{AP}, S_i, s_i^0, M_i, F_i \rangle$  recognizes  $R_i$ , then an appropriate satellite is  $\mathcal{I} = \langle 2^{AP}, 2^{\{\bullet_1, \dots, \bullet_k\}}, S, s^0, M, i_1 \rangle$ , where  $S = S_1 \times S_2 \times \dots \times S_k$ ,  $s^0 = \langle M_1(s_1^0, \tau(v_0)), \dots, M_k(s_k^0, \tau(v_0)) \rangle$ , the transitions are as in a usual product of automata, and for every  $\langle s_1, s_2, \dots, s_k \rangle \in S$  and  $1 \leq i \leq k$ , we have that  $\bullet_i \in i_1(\langle s_1, s_2, \dots, s_k \rangle)$  iff  $s_i \in F_i$ .

**A clock.** A clock notifies PLAYER 1 how many vertices of PLAYER 2 are visited between visits in her own vertices. This is done by a behavioral satellite for the regular language  $R = (2^{AP})^*$ . Indeed, then, PLAYER 1 is notified in every step.

## 6 Complexity for the Different Satellites

Recall that the complexity of deciding a game depends on the size of the satellite. Formally, for a satellite  $\mathcal{I} = \langle O, I, S, s_0, M, i_1, i_2 \rangle$ , the state space of the NBT whose nonemptiness we check in Theorem 5 is a product of  $S$  with other parameters. In this section we study the size of different satellites, and the way it affects the complexity.

We start with structural satellites. It is easy to see that the structural satellites described in Section 5.1 are such that  $S = V$  or  $S = V \times C$ , for some constant set  $C$ . Moreover, since the satellite follows the play (formally, in all states of the UCT constructed in Theorem 4,

the  $V$ -component agrees with the  $V$ -component of  $S$ . Accordingly, we don't need the  $V$ -component in the state space and can maintain  $C$  only. In other words, the state space of  $\mathcal{A}_G$  can be redefined as  $V \times Q \times C \times \{\perp, \top\}$ , and the complexity of the decision problem is reduced accordingly.

We continue to simple behavioral satellites. One is the clock from Section 5.2, which involves a satellite with a single state, leading to  $\mathcal{A}_G$  with state space  $V \times Q \times \{\perp, \top\}$ , and a simpler definition of updated objectives. Another easy special case are *propositional satellites*, which notify PLAYER 1 whenever the play visits a vertex  $v$  such that  $\tau(v) \models \theta$ , for an assertion  $\theta$  over  $AP$ . Indeed, for such notifications we need a two-state satellite. We note that in both cases, EXPTIME-hardness of the game is valid. While the case of propositional satellites this follows by an easy reduction from the case of perspective games with no notifications, for the case of clocks such a reduction is impossible. Nevertheless, since the game constructed in the reduction in the lower-bound proof in [8] alternates between  $V_1$  and  $V_2$ , the result applies also in the clock setting.

Our focus in this section is general behavioral satellites. Consider a regular language  $R$ . We distinguish between the case where the satellite notifies PLAYER 1 whenever the computation since the beginning of the game is in  $R$  (termed *single-track* satellites, as they follow a single computation), and the case where the satellite notifies PLAYER 1 whenever a suffix of the computation is in  $R$ , or equivalently, whenever the computation is in  $\text{true}^* \cdot R$  (termed *multi-track* satellites, as they follow all suffixes of the computation). Analyzing the complexity of games with behavioral satellites, we assume a game is given by a tuple  $\mathcal{G} = \langle G, A_R, \mathcal{U}, t \rangle$ , where  $G$  and  $\mathcal{U}$  are the game graph and winning condition,  $A_R$  is the *pattern automata*, namely the automata describing a regular property  $R$ , and  $t \in \{\text{SINGLE}, \text{MULTI}\}$ , is a flag indicating whether the satellite is single- or multi-track.

► **Theorem 7.** *Deciding whether PLAYER 1  $P$ -wins in a game  $\mathcal{G} = \langle G, A_R, \mathcal{U}, t \rangle$  can be solved in time polynomial in  $|G|$ , exponential in  $|\mathcal{U}|$ , and*

- *polynomial in  $|A_R|$  when  $t = \text{SINGLE}$  and  $A_R$  is a DFW.*
- *exponential in  $|A_R|$  when  $t = \text{MULTI}$  or  $A_R$  is an NFW. Moreover, the problem is EXPTIME-complete already for a fixed-size  $\mathcal{U}$ .*

**Proof.** The upper bounds follow from Theorem 5, and the fact we can generate from  $A_R$  a satellite with no blow-up when  $t = \text{SINGLE}$  and  $A_R$  is a DFW, and a satellite exponential in  $A_R$  when  $t = \text{MULTI}$  or  $A_R$  is an NFW. Note that when  $t = \text{MULTI}$ , we first add to  $A_R$  a  $\text{true}^*$  self-loop leading to the initial state, which makes it nondeterministic.

We continue to the EXPTIME lower bound, and start with the case  $t = \text{SINGLE}$  and  $A_R$  is an NFW. We describe a reduction from linear-space alternating Turing machines (ATM). An ATM is a tuple  $M = \langle Q_e, Q_u, \Gamma, \Delta, q_{init}, q_{acc}, q_{rej} \rangle$ , where  $\Gamma$  is the alphabet,  $Q_e$  and  $Q_u$  are finite sets of *existential* and *universal* states, and we let  $Q = Q_e \cup Q_u$ . Then,  $q_{init}$ ,  $q_{acc}$ , and  $q_{rej}$  are the initial, accepting, and rejecting states, respectively. In the membership problem, we get as input an ATM  $M$  and a word  $w \in \Gamma^*$ , and we decide whether  $M$  accepts  $w$ . The membership problem is EXPTIME-hard already for  $M$  of a fixed size, and when  $\Delta$  has a binary branching degree and alternates between existential and universal states, Thus,  $\Delta \subseteq (Q_e \times \Gamma \times Q_u \times \Gamma \times \{L, R\}) \cup (Q_u \times \Gamma \times Q_e \times \Gamma \times \{L, R\})$ .

A configuration of  $M$  on  $w = w_1, \dots, w_n$  describes its state, the content of the working tape, and the location of the reading head. Assume  $s : \mathbb{N} \rightarrow \mathbb{N}$  is a linear function such that the number of cells used by the working tape in every configuration of  $M$  on its run on  $w$  is bounded by  $s(n)$ . We encode a configuration of  $M$  by a string  $\# \gamma_1 \gamma_2 \dots (q, \gamma_i) \dots \gamma_{s(n)}$ . That is, a configuration starts with  $\#$ , and all its other letters are in  $\Gamma$ , except for one letter

in  $Q \times \Gamma$ . Then,  $M$  is in state  $q$ , the content of the  $j$ -th tape cell is  $\gamma_j$ , and the reading head points at cell  $i$ . We say that the configuration is *existential* if  $q \in Q_e$  and that it is *universal* if  $q \in Q_u$ . The initial configuration of  $M$  on  $w$ , is then  $\#(q_{init}, w_1) \cdot \dots \cdot w_n \cdot \sqcup^{s(n)-n}$ , for the special letter  $\sqcup \in \Gamma$ . We also assume that the initial configuration is existential. If the current state is  $q_{acc}$  or  $q_{rej}$ , then the configuration is final and has no successors. Otherwise, the successors of a configuration  $\#\gamma_1\gamma_2\dots(q, \gamma_i), \dots, \gamma_{s(n)}$  are determined by  $\Delta$ .

Given an ATM  $M$  and a word  $w \in \Gamma^*$ , we construct a game  $\mathcal{G} = \langle G, A_R, \mathcal{U}, \text{SINGLE} \rangle$  such that PLAYER 1 P-wins  $\mathcal{G}$  iff  $M$  accepts  $w$ . The size of  $\mathcal{U}$  is fixed, and  $G$  and  $A_R$  are of size linear in  $s(n)$ , for  $n = |w|$ . The details of the reduction can be found in the full version. Below we describe the key ideas in it.

Essentially, PLAYER 1 generates a legal accepting computation in the computation tree of  $M$  on  $w$ . Thus PLAYER 1 chooses successors in existential configurations, and PLAYER 2 chooses successors in universal ones. The challenging part of the reduction is to guarantee that the sequence of configurations generated is a legal computation, and to do it with a fixed size winning condition. Recall that we encode a configuration of  $M$  by a string  $\#\gamma_1\gamma_2\dots(q, \gamma_i)\dots\gamma_{s(n)}$ . When  $\mathcal{U}$  is polynomial, it is easy to relate letters in the same address in successive configurations, making sure that the transition function of  $M$  is respected. When  $\mathcal{U}$  is of a fixed size, it is not clear how to do it, as such letters are  $s(n)$ -letters apart. The key idea is to use  $A_R$  in order to do the required counting: We let PLAYER 2 choose an address  $k \in \{1, \dots, s(n)\}$  and challenge PLAYER 1 by raising a flag whenever the address is  $k$ . The winning condition  $\mathcal{U}$  checks that the transition function of  $M$  is respected whenever the flag is raised, which forces PLAYER 1 to respect the transitions function of  $M$  in address  $k$ . Moreover, since PLAYER 1 does not know  $k$ , she has to always respect the transition function. The above mechanism is not sufficient, as PLAYER 2 may try to fail PLAYER 1 by raising the flag maliciously, that is, not sticking to one address  $k$ . This is where the notifications enter the picture: the language  $R$  detects malicious flag raises and notifies PLAYER 1 about them. For this,  $A_R$  has to count to  $s(n)$ , but this is allowed, and enables  $\mathcal{U}$  to skip the counting. In addition,  $\mathcal{U}$  restricts the check of PLAYER 1 only to ones in which the flag is raised properly.

Then, when  $t = \text{MULTI}$  and  $A_R$  is a DFW (or NFW), the reduction is similar and is based on the fact that the only nondeterminism in  $A_R$  above is in guessing malicious flag raises, namely raises that are not  $s(n)$  letters apart. Such a behavior can be specified by a regular expression  $\text{true}^* \cdot R$  for  $R$  that can be described by a DFW of size polynomial in  $s(n)$ . ◀

---

## References

- 1 S. Agarwal, M. S. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. 32nd IEEE International Conference on Computer Communications*, pages 2211–2219, 2013.
- 2 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
- 3 D. Berwanger, K. Chatterjee, M. De Wulf, L. Doyen, and T. A. Henzinger. Strategy construction for parity games with imperfect information. *Information and Computation*, 208(10):1206–1220, 2010.
- 4 R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.
- 5 K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. 16th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, pages 1–14. Springer, 2010.
- 6 K. Chatterjee, L. Doyen, T. A. Henzinger, and J-F. Raskin. Algorithms for  $\omega$ -regular games with imperfect information. In *Proc. 15th Annual Conf. of the European Association for*

- Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302, 2006.
- 7 D. Fisman and O. Kupferman. Reasoning about finite-state switched systems. In *5th International Haifa Verification Conference*, volume 6405 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2009.
  - 8 O. Kupferman and G. Vardi. Perspective games. In *Proc. 34th IEEE Symp. on Logic in Computer Science*, pages 1–13, 2019.
  - 9 O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
  - 10 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.
  - 11 D. Liberzon. *Switching in Systems and Control*. Birkhauser, 2003.
  - 12 Y. Lustig and M.Y. Vardi. Synthesis from component libraries. *Software Tools for Technology Transfer*, 15(5-6):603–618, 2013.
  - 13 M. Margaliot. Stability analysis of switched systems using variational principles: an introduction. *Automatica*, 42(12):2059–2077, 2006.
  - 14 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.
  - 15 B. Puchala. Asynchronous omega-regular games with partial information. In *35th Int. Symp. on Mathematical Foundations of Computer Science*, pages 592–603. Springer, 2010.
  - 16 J.H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science*, 29:274–301, 1984.
  - 17 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

# On the Complexity of Multi-Pushdown Games

Roland Meyer

TU Braunschweig, Germany  
roland.meyer@tu-bs.de

Sören van der Wall

TU Braunschweig, Germany  
s.van-der-wall@tu-bs.de

---

## Abstract

We study the influence of parameters like the number of contexts, phases, and stacks on the complexity of solving parity games over concurrent recursive programs. Our first result shows that  $k$ -context games are  $b$ -EXPTIME-complete, where  $b = \max\{k-2, 1\}$ . This means up to three contexts do not increase the complexity over an analysis for the sequential case. Our second result shows that for ordered  $k$ -stack as well as  $k$ -phase games the complexity jumps to  $k$ -EXPTIME-complete.

**2012 ACM Subject Classification** Theory of computation → Automata over infinite objects

**Keywords and phrases** concurrency, complexity, games, infinite state, multi-pushdown

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.52

**Funding** This work was partially supported by DFG grant 417532197 *Effective Denotational Semantics for Synthesis (EDS@SYN)*.

## 1 Introduction

Software verification and synthesis are difficult, even more so when concurrency comes into play. Algorithmically, both tasks often amount to solving games [33] over an operational model that captures implementation and specification details [62, 40]. What makes these games hard to solve is the size of the underlying graph, which easily ends up having an infinite set of positions. One reason is that software often computes over infinite data domains. Another reason is that the control flow tends to be structured into recursive procedures or even functional code. Despite this difficulty, efficient algorithms and tools for solving games over infinite graphs have been proposed. Data aspects are discharged to logical reasoning engines [9, 21]. Recursive functions are summarized to their call-return relationship [58, 54, 61, 10], an idea that generalizes to functional programs [3, 46, 45, 55, 34, 41, 36]. Alternatively, the set of reachable call stacks is tracked symbolically and saturated until a fixed point is reached [20, 14, 32, 23], which is again applicable to functional programs [15, 37, 26, 17, 25]. There are efficient implementations of saturation [18, 19]. Yet, tools that participate in the Software Verification Competition [11], like CPACHECKER [1, 12, 13] and the ULTIMATE framework [2, 38, 39], favor summarization.

What remains a challenge, not only for game solvers but already for verification engines, is concurrency. When combined with recursion, even the simplest analysis problems become undecidable [31, 53]. One way out is under-approximation, analyzing only a (critical) subset of the semantics. In *context-bounded* computations [52] the thread holding the processor (the context) may switch only a bounded number of times. *Phase-bounded* computations [47] generalize the idea. During a phase all threads may push their stack but only one thread can pop. In *ordered* computations [16], the threads are ordered and a pop transition may only be performed by the smallest thread whose stack is non-empty. The complexity is similar to the phase-bounded case [6, 5]. Technically, the above results are obtained for multi-pushdown systems [16], a programming model with multiple stacks accessed by a sequential control flow representing the interleaving of the threads.



© Roland Meyer and Sören van der Wall;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 52; pp. 52:1–52:35



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Overview of the state-of-the-art and new results with technical highlights marked.

	Previous results		New results	
	$k$ fixed	$k$ input	$k$ fixed	$k$ input
<b>Upper bounds</b>				
$k$ -stack ordered	—	—	$k$ -EXP	non-elem.
$k$ -context	$k$ -EXP [57]	non-elem. [57]	$\max\{k - 2, 1\}$ -EXP	non-elem.
$k$ -phase	$k$ -EXP [57]	non-elem. [57]	$k$ -EXP	non-elem.
<b>Lower bounds</b>				
$k$ -stack ordered	—	—	$k$ -EXP	non-elem.
$k$ -context	—	—	$\max\{k - 2, 1\}$ -EXP	non-elem.
$k$ -phase	—	non-elem. [7, 8]	$k$ -EXP	non-elem.

The aforementioned works are limited to (linear-time) verification. There are considerably less results towards under-approximate synthesis (and branching-time model checking). Seth was the first to study parity games over multi-pushdown systems, multi-pushdown games (MPDG) for short [57]. He considered phase boundedness and gave a summarization-based decision procedure. It can be lifted to a subclass of concurrent higher-order programs [56]. Using saturation, Hague [35] was able to capture the full class of concurrent higher-order programs. The algorithm works for orderedness, bounded phases, and the bounded scopes explained below. The winning condition is reachability. Also using saturation, Atig et al. [8] showed how to reduce the number of phases in an MPDG, leading to a recursive decision procedure. It yields a  $k$ -EXPTIME upper bound for  $k$  phases. If the phases are part of the input, the upper bound is non-elementary and the authors present a matching lower bound. An also non-elementary lower bound was shown for the related problem of branching-time model checking under a given context bound [7].

**Contribution.** We determine the precise influence of the number of contexts, phases, and stacks on the complexity of solving parity MPDG (Table 1). The practically most relevant and at the same time technically most interesting case is  $k$ -context parity MPDG for which we show  $\max\{k - 2, 1\}$ -EXPTIME-completeness.<sup>1</sup> The upper bound reflects the fact that three contexts can be translated into a single stack pushdown. Interestingly, each further context increases the complexity by one exponent, as in the case of the seemingly more expressive phase-bounded MPDG. There, the complexity settles at  $k$ -EXPTIME-complete for  $k$  phases. The same complexity holds for ordered  $k$ -stack parity MPDG.

Motivated by the success of summarization algorithms [1, 13, 2, 38], we decided to derive our upper bounds by summarization. The algorithms reduce the given MPDG to a finite game by abstracting plays through completed function calls (between matching pushes and pops) to their effect on the control states. This approach has been pioneered by Walukiewicz for pushdown games [61] and generalized to phase-bounded MPDG by Seth [57]. Our algorithms are generalizations and optimizations of Seth’s work. Unlike [57], we do not assume a constant number of stacks under context bounds.

Unfortunately, we also discovered a flaw in [57] that makes the existing finite game construction unsound. We explain and fix the problem, and thus obtain the first (correct) summarization algorithm for MPDG.

We complement the findings by matching lower bounds. They work by reductions from space-bounded alternating Turing machines. To demonstrate the expressiveness of MPDG without getting lost in the case distinctions often involved with Turing machine reductions,

<sup>1</sup> Class  $k$ -EXPTIME is the union of all  $\text{DTIME}(\exp_k(\text{poly}(n)))$  with  $\exp_0(n) = n$ ,  $\exp_{k+1}(n) = 2^{\exp_k(n)}$ .

we propose the formalism of first-order relations. These are relations among words formulated in a fragment of first-order logic. Our main result shows that  $k$ -phase,  $(k + 2)$ -context, and  $k$ -stack ordered MPDG can decide first-order relations over words of length  $(k - 1)$ -fold exponential. Reachability is sufficient as the winning condition. The reductions are a considerable step beyond the non-elementary and parameterwise rough lower bounds in [7, 8]. We build on ideas in [7], the result for phase-bounded MPDG in [8] cannot be adapted to context boundedness.

**Related Work.** There are further restrictions on concurrent recursive programs. Round-bounded computations [48] schedule the threads in round-robin fashion for a given number of rounds. Scope-bounded computations [49] require a matching pop to occur within a bounded number of contexts from the corresponding push. Very recently, hole boundedness [4] has been proposed as a generalization of bounded scope. Common to these notions is that they limit the ability of the scheduler in contrast to the studied restrictions.

A framework that has led to algorithmic meta-theorems of the above form is bounded tree-width [28, 50] and its developments like split-width [29, 22]. The idea is to capture a programming model that acts on infinite storage by a finite-state device operating over enriched computations. So far, this approach has not been lifted to games.

Remotely related is higher-order model checking [51]. The complexity is similar, namely  $k$ -EXPTIME for schemes of order  $k$ . Moreover, besides Ong's game semantics approach [51], there are saturation [17] and summarization [46] algorithms. The technical challenges, however, are different. In HOMC, the task is to represent and manipulate recursively defined functions of higher order. In MPDG, the task is to capture the interferences among threads.

## 2 Multi-Pushdown Games

**Multi-Pushdown Systems.** A *multi-pushdown system (MPDS)* is a finite-control program that operates on finitely many stacks of unbounded height [16]. Formally, it is a tuple  $P = (Q, \Gamma, \delta, n)$ , where  $Q$  is a finite set of control states,  $n$  is the number of stacks,  $\Gamma$  is a finite stack alphabet, and  $\delta = \delta_{int} \cup \delta_{push} \cup \delta_{pop}$  is a set of internal, push, and pop transitions with

$$\delta_{int} \subseteq Q \times [1..n] \times Q \quad \delta_{push} \subseteq Q \times [1..n] \times \Gamma \times Q \quad \delta_{pop} \subseteq Q \times \Gamma \times [1..n] \times Q.$$

Each transition acts on a stack  $r \in [1..n]$ . We refer to all internal transitions for stack  $r$  with  $\delta_{int,r}$ , and similarly for  $\delta_{push,r}$  and  $\delta_{pop,r}$ . Let  $\delta_r = \delta_{int,r} \cup \delta_{push,r} \cup \delta_{pop,r}$ . The size of  $P$  is given by  $|P| = |Q| + |\Gamma| + |\delta|$ .

The behavior of MPDS is defined in terms of configurations and labeled transitions between them. A configuration of  $P$  is a pair  $(q, \mathcal{P})$  consisting of a control state  $q \in Q$  and a vector of stack contents  $\mathcal{P} \in (\Gamma^*)^n$ . We use  $C = Q \times (\Gamma^*)^n$  for the set of all configurations. The labeled transition relation  $\rightarrow \subseteq C \times \delta \times C$  implements the transitions given by  $P$  on its configurations. We have  $(q, \mathcal{P}) \xrightarrow{\tau} (q', \mathcal{P}')$  if one of the following holds:

$$\begin{array}{lll} \tau = (q, r, q') \in \delta_{int} & \text{and} & \mathcal{P}' = \mathcal{P} \\ \tau = (q, r, s, q') \in \delta_{push} & \text{and} & \mathcal{P}' = \mathcal{P}_{[r \rightarrow s. \mathcal{P}[r]]} \\ \tau = (q, s, r, q') \in \delta_{pop} & \text{and} & \mathcal{P}'_{[r \rightarrow s. \mathcal{P}'[r]]} = \mathcal{P}. \end{array}$$

If transition label  $\tau$  is not important, we may omit it. Vector  $\mathcal{P}_{[j \rightarrow y]}$  is defined to coincide with  $\mathcal{P}$  except for the content of stack  $j$  which is replaced by  $y$ ,  $\mathcal{P}_{[j \rightarrow y]}[j] = y$  and  $\mathcal{P}_{[j \rightarrow y]}[z] = \mathcal{P}[z]$  for all  $z \neq j$ . We use  $\mathcal{P}[1..r] = y$  to indicate that stacks 1 to  $r$  hold content  $y$ .

**Ordered Computations, Contexts, and Phases.** A computation of  $P$  is a finite or infinite sequence of configurations  $(q_0, \mathcal{P}_0) \xrightarrow{\tau_0} (q_1, \mathcal{P}_1) \xrightarrow{\tau_1} \dots$  that respects the transition relation. It is *ordered* [16] if for every pop transition from stack  $r$  the stacks  $1, \dots, r-1$  are empty,  $\mathcal{P}_p[1..r-1] = \varepsilon$  for all  $\tau_p \in \delta_{pop,r}$ . The computation is a *context on stack  $r$*  if all transitions act on that stack, for all  $\tau_p$  we have  $\tau_p \in \delta_r$ . It is a *phase on stack  $r$*  if every pop transition acts on that stack, for all  $\tau_p \in \delta_{pop}$  we have  $\tau_p \in \delta_{pop,r}$ . A computation is said to have  $k$  *contexts* [52] if it decomposes into  $k$  contexts but does not decompose into  $k-1$  contexts, and similar for  $k$  *phases* [47].

**Graph Games.** A *graph game* is a two-player zero-sum game played by moving a pebble along the edges of a potentially infinite graph [33]. Formally, it is a tuple  $(V, E, own, win)$ , where  $(V, E)$  is a directed graph,  $own : V \rightarrow \{\text{Eve}, \text{Ana}\}$  is an ownership function, and  $win : V^\omega \rightarrow \{\text{Eve}, \text{Ana}\}$  is a winning condition. We call  $V$  the positions and  $E$  the moves of the game. We call a graph game finite, if the set of positions is finite.

A *play* is a maximal path  $\pi$  in the graph  $(V, E)$  underlying the game. *Eve wins the play* if either the play is infinite and  $win(\pi) = \text{Eve}$ , or the play is finite and ends in a position from  $V_{\text{Ana}}$  (with no move left for Ana). Otherwise, *Ana wins the play*. Whenever a play reaches a position, the owner of the position has to decide about the next move. A *strategy for Eve* is a function  $\sigma : V^*V_{\text{Eve}} \rightarrow V$  such that  $v E \sigma(\pi v)$  holds for all  $\pi v \in V^*V_{\text{Eve}}$ . The strategy is *positional* if it only depends on the current position. In this case, the strategy can be given as  $\sigma : V_{\text{Eve}} \rightarrow V$ . A play  $\pi = \pi_0\pi_1\dots$  is *compliant with strategy  $\sigma$*  for Eve if for all  $\pi_p \in V_{\text{Eve}}$  we have  $\pi_{p+1} = \sigma(\pi_0\dots\pi_p)$ . A strategy for Eve is *winning from position  $v$*  if Eve wins all compliant plays that start in  $v$ . If there is a strategy that is winning from  $v$ , we call  $v$  a *winning position*. The definitions for Ana are similar.

A *reachability winning condition*  $win_W$  is defined by a set  $W \subseteq V$  of so-called winning positions. We define  $win_W(\pi) = \text{Eve}$  if a winning position  $\pi_i \in W$  is visited, and  $win_W(\pi) = \text{Ana}$  otherwise. A *reachability game* is a tuple  $(V, E, own, W)$ . A *parity winning condition*  $win_\Omega$  is defined by a mapping from positions to *priorities*,  $\Omega : V \rightarrow [0..max]$ . The winner of a play is determined by the highest priority that occurs infinitely often during the play, i.e.  $win_\Omega(\pi) = \text{Eve}$  if the highest infinitely often occurring priority is even, and  $win_\Omega(\pi) = \text{Ana}$  if it is odd. A *parity game* is a tuple  $(V, E, own, \Omega)$ .

**Multi-Pushdown Games.** An  $n$ -stack *multi-pushdown game (MPDG)* [57] is a triple of the form  $G = (P, own, win)$  consisting of a multi-pushdown system  $P = (Q, \Gamma, \delta, n)$ , an ownership function  $own : Q \rightarrow \{\text{Eve}, \text{Ana}\}$ , and a winning condition  $win$ . The MPDG induces the graph game  $(C, \rightarrow, own, win)$ , and we say that Eve wins the MPDG if she wins the induced graph game. The set of positions  $C$  and the set of moves  $\rightarrow$  are the configurations and transition relation of  $P$  as defined above. The ownership function carries over from control states to configurations,  $own(q, \mathcal{P}) = own(q)$  for all  $(q, \mathcal{P}) \in C$ . To be precise, in a *reachability MPDG* we are given a set of control states  $Q_{reach}$  to represent the winning set  $C_{reach} = Q_{reach} \times (\Gamma^*)^n$ . In a *parity MPDG*, the winning condition is given by a priority assignment  $\Omega : Q \rightarrow [0..max]$  to the states. Again, we lift it to configurations by  $\Omega(q, \mathcal{P}) = \Omega(q)$ . The size of a reachability MPDG  $G$  is  $|G| = |P|$ , the size of a parity MPDG is  $|G| = |P| + max$ .

A  $k$ -context *multi-pushdown game*  $(G, k)$  is a restriction of the MPDG  $G$  so that plays have at most  $k$  contexts [7]. The formal definition tracks the number of contexts within the positions. Moves that would introduce context  $k+1$  do not exist. The definition of  $k$ -phase *multi-pushdown games* is similar [57, 8]. An *ordered  $n$ -stack multi-pushdown game* only admits moves that lead to an ordered play: a pop transition on stack  $r$  exists only in positions where stacks  $1$  to  $r-1$  are empty.

In our development, it will be convenient to assume that the MPDG of interest does not deadlock. Every MPDG  $G$  can be turned into a deadlock-free MPDG  $G'$  that has the same winner, the same highest priority, and is larger by only a linear factor. This continues to hold with a polynomial factor under the aforementioned restrictions.

### 3 Upper Bound for Ordered MPDG

We give an algorithm for computing the winning positions in an ordered  $n$ -stack MPDG that works in  $n$ -EXPTIME. The algorithm is a slight optimization of Seth's summarization construction for phase-bounded MPDG [57]. We discovered a bug in this construction that we explain and show how to correct. Details can be found in Appendix B.

► **Theorem 1.** *Given an ordered  $n$ -stack MPDG  $G$  with parity winning condition, we can compute Eve's winning positions of the form  $(q, \varepsilon^n)$  in time  $\exp_n(\text{poly}(|G|^2))$ .*

The algorithm constructs from the given MPDG  $G$  a finite parity game  $F$  and solves the latter. The finite game preserves the winner for the positions of interest, the set of priorities, and is not too costly to compute. For the complexity, note that  $n$  is not part of the input but fixed. Further, parity games are solved in time exponential only in the number of priorities [42, 43].

► **Lemma 2.** *Let  $G$  be an ordered  $n$ -stack parity MPDG. In time  $\exp_n(\text{poly}(|G|^2))$  we can compute a finite parity game  $F$  so that Eve wins  $G$  from position  $(q, \varepsilon^n)$  if and only if she wins  $F$  from a corresponding position. The priorities in  $G$  and  $F$  coincide.*

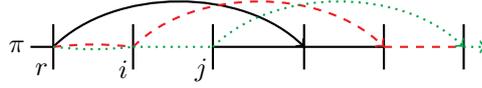
#### 3.1 Summarization for Ordered MPDG

We explain the summarization construction from [57], highlight our optimization for ordered MPDG, and finally make the construction formal. The game  $G$  has infinitely many position due to arbitrarily growing stacks. The finite parity game  $F$  removes the stacks and instead tracks the current top of stack symbol for each stack. This forbids pop transitions in  $F$ . When one player decides to make a pop transition  $F$  ends and determines a winner.

In order to model  $G$ 's behavior after a pop,  $F$  implements a summarization mechanism. When a symbol  $s$  is pushed onto a stack  $r$ , Eve proposes a set of summaries. This set can be understood as fixing a strategy for Eve in  $G$  that she will follow for as long as  $s$  remains on stack  $r$ . Fixing a strategy results in the set of all plays that are compliant to it and lead from the push of  $s$  to a situation where  $s$  is popped again. Each summary captures such a situation and thereby abstracts a play from this set. The finite game  $F$  can thus skip any of the abstracted plays up to the captured situation after the pop of  $s$ .

There is no guarantee that Eve will be honest in the sense that the proposed set of summaries indeed abstracts all plays that pop  $s$  and are compliant to some fixed strategy. To account for this, Ana is allowed to react to the proposal. First, she may trust Eve by choosing a summary from the proposed set. In this case,  $F$  executes the skip of the abstracted play and replaces (parts of) the current position with the position after the pop as captured by the summary. This can be understood as also fixing a strategy for Ana in  $G$  that, together with Eve's strategy, leads to the abstracted play.

Second, she may doubt Eve's proposal by executing the push transition instead of skipping. This replaces the top of stack symbol for stack  $r$ . Executing the push also stores the set of summaries proposed by Eve in the position of  $F$ . It is remembered for as long as  $s$  remains the topmost symbol of stack  $r$ . To be precise, the position will hold a separate set of



■ **Figure 1** A fixed play  $\pi$  in  $G$ . Each arc matches a push to its pop. The highlighted paths show different plays in  $F$ .

summaries for the topmost symbol of each stack. When a stack is popped, the remembered set of summaries for that stack is checked for containing a summary that captures the current situation of the play. Eve wins if and only if some summary in the set applies.

Finally, there may be another reason for Ana to execute the push transition. In this case, she trusts Eve's proposed set, but her own strategy will make sure that  $s$  is never removed from the stack.

**Ordered Summaries.** To abstract a play  $\pi$  in  $G$  from a push to a matching pop on a stack  $r$ , a summary for an ordered MPDG, or ordered summary for short, takes the shape

$$(q, m, \mathcal{T}, \mathcal{M}, \mathcal{S}).$$

The entries  $q \in Q$  and  $\mathcal{T} \in \Gamma^{n-r}$  describe the configuration resulting from the final pop transition, with  $q$  the control state and  $\mathcal{T}$  the topmost symbol for each stack. The orderedness restriction forces the stacks  $1, \dots, r-1$  to be empty. The entry  $m \in [0..max]$  is the maximal priority encountered during the abstracted play, from after the push up to before the pop. Each entry  $\mathcal{M}[j]$  of  $\mathcal{M} \in [0..max]^{n-r}$  is the maximal priority encountered since after the push of the topmost symbol  $\mathcal{T}[j]$  of stack  $j$ .

The summary recursively holds a vector  $\mathcal{S} \in (2^{\mathbb{OS}})^{n-r}$  of sets of summaries for the other non-empty stacks. Here,  $\mathbb{OS}$  is the set of all summaries as defined below. Assume the abstracted play pushes the symbol  $\mathcal{T}[j]$  onto a stack  $j \neq r$  and does not contain a matching pop. The set of summaries  $\mathcal{S}[j]$  is Eve's proposed set for how the top of stack symbol  $\mathcal{T}[j]$  can be popped after the abstracted play. When the play skips to the position captured by this summary,  $\mathcal{S}[j]$  becomes the set of summaries stored for  $\mathcal{T}[j]$ .

**Pathing.** When Ana doubts a set of summaries for the push on stack  $r$  she might have a strategy that pops stack  $r$  with a combination of control state, highest priority and top of stack symbols, that are not present in the set. Alternatively, they coincide but after the pop on stack  $r$ , her strategy pops stack  $j$  in a situation not captured by  $\mathcal{S}[j]$ . Observe that if Ana wants to doubt a set  $\mathcal{S}[j]$ , she needs to find a play in  $F$ , which runs into a pop on stack  $j$  without running into a pop on another stack. Alternatively, if she wants to skip to a situation captured by a summary within  $\mathcal{S}[j]$ , she needs to steer the play to run into the push of  $\mathcal{T}[j]$ , so she gets the option of skipping.

To understand this, assume some summary abstracts a play  $\pi$  in  $G$  from the push to a pop on stack  $r$ , which first contains a push on stack  $i$  and then a push on stack  $j$  (Figure 1). If the set of summaries  $\mathcal{S}[j]$  does not capture the situation how  $\pi$  pops  $\mathcal{T}[j]$ , Ana can run into its pop as illustrated by the dashed path in the figure, i.e. Ana executes the push on stack  $r$  and then skips upon the push on stack  $i$ . The play reaches the pop on stack  $j$  and is checked against  $\mathcal{S}[j]$ . If it is captured, Ana wants to continue the play beyond the pop of stack  $j$  by skipping to a summary in  $\mathcal{S}[j]$ . To achieve this, she executes both, the push on stack  $r$  and  $i$ . Then she can skip upon reaching the push on stack  $j$  as illustrated by the dotted path.

► **Definition 3.** We proceed by induction on the stack from  $n$  down to one. The set of ordered summaries for stack  $r$  is

$$\mathbb{OS}_r = Q \times [0..max] \times \Gamma^{n-r} \times [0..max]^{n-r} \times \prod_{r < j \leq n} 2^{\mathbb{OS}_j}.$$

In the base case  $r = n$ , the last three components are defined to be absent. The set of all ordered summaries is  $\mathbb{OS} = \bigcup_{1 \leq r \leq n} \mathbb{OS}_r$ .

Our optimization targets the orderedness restriction. During an abstracted play between a push and a pop on stack  $r$ , if we find an unmatched push on another stack  $j$ , then we can conclude that  $j > r$ . This means a summary for stack  $r$  only needs to contain summaries for stacks of larger order. The largest set is  $\mathbb{OS}_1$  which has size  $exp_{n-1}(\mathcal{O}(|G|^2))$ , Appendix A.1. When we construct the game  $F$ , it will be convenient to assume that all vectors have length  $n$ . We fill the missing entries for stacks one to  $r$  with  $\varepsilon$  for  $\mathcal{T}$ , 0 for  $\mathcal{M}$ , and  $\emptyset$  for  $\mathcal{S}$ .

### 3.2 The Finite Parity Game

Consider the ordered  $n$ -stack MPDG with parity winning condition  $G = (P, own, \Omega)$ , where  $P = (Q, \Gamma, \delta, n)$ ,  $own : Q \rightarrow \{\text{Eve, Ana}\}$ , and  $\Omega : Q \rightarrow [0..max]$ . We define the finite parity game  $F$  explained above, following Seth [57] but correcting a mistake. Rather than giving the positions of  $F$  right away, we explain the behavior of the game and introduce them together with their moves. Game  $F$  regularly visits check positions  $(\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$  with  $q \in Q$ ,  $\mathcal{T} \in \Gamma^n$ ,  $\mathcal{M} \in [0..max]^n$ , and  $\mathcal{S} \in (2^{\mathbb{OS}})^n$ . Note that there is a set of summaries for each stack. The owner and the priority are the ones for  $q$ .

**Internal Transitions.** Internal transitions  $(q, r, p) \in \delta_{int}$  of game  $G$  are mirrored in  $F$ . They only update the priorities:

$$(\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \rightarrow (\text{Check}, p, \mathcal{T}, upd(\mathcal{M}, p), \mathcal{S}). \quad (1)$$

The new priority vector is defined by  $upd(\mathcal{M}, q)[j] = \max\{\mathcal{M}[j], \Omega(q)\}$ , for each stack  $j$ . Note that we use an implicit universal quantification over the parameters that are not specified further, meaning the transition exists for all  $\mathcal{T}$ ,  $\mathcal{M}$ , and  $\mathcal{S}$ .

**Push Transitions.** Push transitions  $(q, r, s, p) \in \delta_{push}$  in  $G$  lead to a series of transitions in  $F$  originating from  $(\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$ :

$$(\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \rightarrow (\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, p, s) \quad (2)$$

$$(\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, p, s) \rightarrow (\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, p, s, \mathcal{S}) \quad (3)$$

$$(\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, p, s, \mathcal{S}) \rightarrow (\text{Check}, p, \mathcal{T}_{[r \rightarrow s]}, upd(\mathcal{M}, p)_{[r \rightarrow \Omega(p)]}, \mathcal{S}_{[r \rightarrow s]}) \quad (4)$$

$$(\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, p, s, \mathcal{S}) \rightarrow (\text{Jump}_r, q', m', \mathcal{T}'', \mathcal{M}', \mathcal{S}'', \mathcal{M}[r]) \quad (5)$$

$$(\text{Jump}_r, q', m', \mathcal{T}'', \mathcal{M}', \mathcal{S}'', \mathcal{M}[r]) \rightarrow (\text{Check}, q', \mathcal{T}'', \mathcal{M}'', \mathcal{S}''). \quad (6)$$

The transitions introduce intermediary push, claim, and jump positions with the following ownership and priority assignments:

$$\begin{aligned} own(\text{Push}_r, -) &= \text{Eve} & \Omega(\text{Push}_r, -) &= 0 \\ own(\text{Claim}_r, -) &= \text{Ana} & \Omega(\text{Claim}_r, -) &= 0 \\ own(\text{Jump}_r, -) &= \text{Eve} & \Omega(\text{Jump}_r, q', m', \mathcal{T}'', \mathcal{M}', \mathcal{S}'', \mathcal{M}[r]) &= m'. \end{aligned}$$

Move (2) remembers control state  $p$  and symbol  $s$  and gives Eve the next move. With Move (3), Eve proposes a set of summaries  $S \subseteq \mathbb{OS}_r$  for the symbol to be pushed. By implicit universal quantification, there is a transition for each such set. Move (4) performs the push: the priority vector takes into account the priority of  $p$  for all stacks. For stack  $r$ ,  $\Omega(p)$  is the highest (and only) priority seen since the push of  $s$ . Move (5) corresponds to a skip and exists for every summary  $(q', m', \mathcal{T}', \mathcal{M}', \mathcal{S}') \in S$ . For stack  $r$ , we preserve the top of stack symbol  $\mathcal{T}[r]$  and the set of summaries  $\mathcal{S}[r]$ . For the other stacks, we use the information given by the summary. Thus,  $\mathcal{T}'' = \mathcal{T}'_{[r \rightarrow \mathcal{T}[r]]}$  and  $\mathcal{S}'' = \mathcal{S}'_{[r \rightarrow \mathcal{S}[r]]}$ . The role of the jump position is to make visible the priority  $m'$  of the summary. In Move (6), we update the priority vector to  $\mathcal{M}''$ . For stack  $r$ , note that  $\mathcal{T}[r]$  remains the topmost symbol after the skip. Hence, the priority assignment has to take into account  $\mathcal{M}[r]$ , the highest priority seen before the skip. Thus,  $\mathcal{M}'' = \text{upd}(\mathcal{M}'_{[r \rightarrow \max\{\mathcal{M}[r], m'\}], q'})$ .

### 3.3 Pop Transitions and a Correction to a Mistake

As defined in [57] Ana may win  $F$  in cases where she does not win  $G$ . We correct the definition and explain the difference to the original formulation. The ordered MPDG  $G$  can only perform a pop  $(q, s, r, p) \in \delta_{pop}$  of symbol  $s$  from stack  $r$  if the stacks 1 to  $r - 1$  are empty. Given the side condition, the finite game  $F$  has simulating moves only in positions  $(\text{Check}, q, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}, \mathcal{S})$  where  $\mathcal{T}[1..r - 1] = \varepsilon$ ,  $\mathcal{M}[1..r - 1] = 0$ , and  $\mathcal{S}[1..r - 1] = \emptyset$ . The simulating moves immediately decide about the winner of the game and take the following shape. The positions  $\text{EveWin}$  and  $\text{AnaWin}$  are winning for Eve resp. Ana. Both are owned by Eve, have self-loops, and  $\text{EveWin}$  has priority 0 while  $\text{AnaWin}$  has priority 1.

$$(\text{Check}, q, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}, \mathcal{S}) \rightarrow \text{EveWin} \quad (7)$$

$$(\text{Check}, q, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}, \mathcal{S}) \rightarrow \text{AnaWin} . \quad (8)$$

Recall that the goal of a pop transition is to check whether Ana caught Eve lying on the proposal of summaries for the popped symbol. If the current position is captured by a summary, Eve was honest and Ana could have found a path to skip after the current pop. Otherwise, Eve was lying, Ana was right in questioning the proposal and wins.

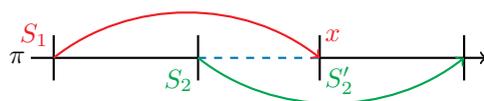
To check whether position  $(\text{Check}, q, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}, \mathcal{S})$  is captured, we compare it to each summary stored for stack  $r$ . The finite game has Move (7) if and only if there is a summary  $x = (p, m, \mathcal{T}', \mathcal{M}', \mathcal{S}') \in \mathcal{S}[r]$  with  $m = \mathcal{M}[r]$ ,  $\mathcal{T}' = \mathcal{T}_{[r \rightarrow \varepsilon]}$ ,  $\mathcal{M}' = \mathcal{M}_{[r \rightarrow 0]}$ , and

$$\mathcal{S}'[j] \subseteq \mathcal{S}[j] \text{ for all } j \neq r. \quad (9)$$

If summary  $x$  exists, it indeed captures the current position in the game: The state after the pop transition is  $p$ , the priority  $m$  is equal the maximal priority seen during the play with  $s$  on stack  $r$ , i.e.  $m = \mathcal{M}[r]$ . The top of stack symbols  $\mathcal{T}'$  coincide with the current ones  $\mathcal{T}$ , also the maximal priorities encountered since the moment these symbols have been pushed coincide,  $\mathcal{M}[j] = \mathcal{M}'[j]$ .

The problem in [57] refers to the relationship between  $\mathcal{S}'$  and  $\mathcal{S}$ . The incorrect definition required an equality and therefore missed Moves (7) winning for Eve. The correction is to require Inclusion (9). To understand the problem with equality, consider the play  $\pi$  in an ordered MPDG  $G$  depicted in Figure 2. The play has two pushes with corresponding pops, one on **stack  $r$**  and drawn **above** the play, the other on **stack  $j$**  and drawn **below** the play. The push on stack  $j$  is simulated in the finite game  $F$  in two different ways.

Upon the push of stack  $r$ , Eve chooses a strategy up until the pop of stack  $r$ , enumerates all compliant plays (up to the pop), and summarizes them in the proposed set  $S_1$ . The play  $\pi$  is among the compliant plays and yields summary  $x \in S_1$ . The part of  $\pi$  abstracted by  $x$



■ **Figure 2** Matching pushes and pops, the **above** on stack  $r$ , the **below** on stack  $j$ .

contains a push on stack  $j \neq r$ . Eve extends her strategy and enumerates all plays from the push to the pop of stack  $j$  that coincide with  $\pi$  on the already fixed *dashed* part, from the *push* of stack  $j$  to the *pop* of stack  $r$ . The resulting set of summaries  $S'_2$  is contained in  $x$ .

Ana may decide against skipping and execute the push on stack  $r$ . The play may follow  $\pi$  and reach the push on stack  $j$ . Eve is again asked to summarize all plays up to the pop on stack  $j$ , and proposes a set  $S_2$ . Even though the proposed sets are for the same push in the same play, the result may be  $S_2 \neq S'_2$ . When forming  $S'_2$ , the play was already fixed on the *dashed* part, up to the pop of stack  $r$ . When forming  $S_2$ , this does not hold. Hence, there may be plays starting with the *push* of stack  $j$  that pop stack  $r$  in a situation not captured by  $x$  and later pop stack  $j$  (e.g. with a different highest priority seen). However, Eve will have to at least propose a summary for each play that coincides with  $\pi$  on the *dashed* part and later pops stack  $j$ . Thus, the formed set of summaries  $S_2$  is a superset of the set  $S'_2$ . Accordingly, if Ana also executes the push on stack  $j$ , and the play runs into the pop of stack  $r$ , checking whether summary  $x$  captures the situation at the pop of stack  $r$  requires that  $S'_2 \subseteq S_2$  and not  $S'_2 = S_2$ .

#### 4 Upper Bound for Context-Bounded MPDG

We give an algorithm to solve context-bounded MPDG that takes  $\max\{1, k - 2\}$ -EXPTIME when considering  $k$  contexts. From a practical point of view, the interesting observation is that communication across two context switches does not increase the complexity over the problem of solving (sequential) pushdown games, which are EXPTIME-complete [61]. This compares well to the fact that a 3-context MPDS can be encoded as a single stack PDS. Interestingly, beyond the third context the complexity rises at the same pace as for phase-bounded MPDG, namely by one exponent per context/phase (Section 5).

► **Theorem 4.** *Given a  $k$ -context MPDG  $G$  with parity winning condition, we can compute Eve's winning positions of the form  $(q, \varepsilon^n)$  in time  $\exp_{\max\{1, k-2\}}(\text{poly}(|G|))$ .*

Note that we do not assume the number of stacks to be fixed. The observation is that in a play with  $k$  contexts we can only make use of  $k$  stacks and it can be converted into an MPDG with only  $k$  stacks at only polynomial overhead (cf. Appendix C).

Our algorithm starts by reducing the number of stacks, if necessary. Afterwards, we construct a finite parity game identical to the one from the previous section except for a different set of summaries. The correctness statement is therefore a variant of Lemma 2, with a similar proof that can be found in [60].

► **Lemma 5.** *Let  $G$  be a  $k$ -context  $k$ -stack parity MPDG. In time  $\exp_{\max\{1, k-2\}}(\text{poly}(|G|))$  we can compute a finite parity game  $F$  so that Eve wins  $G$  from position  $(q, \varepsilon^n)$  if and only if she wins  $F$  from a corresponding position. The priorities in  $G$  and  $F$  coincide.*

The set of summaries we use to construct  $F$  is an optimization of Seth's summaries for phase-bounded MPDG. We repeat Seth's definition in our notation.

## 52:10 On the Complexity of Multi-Pushdown Games

► **Definition 6** ([57]). We fix a stack  $r$  and define the set of phase summaries  $\mathbb{PS}_{r,c}$  by induction on the phase  $c$  from  $k$  down to 1:

$$\mathbb{PS}_{r,c} = Q \times \{c\} \times [1..max] \times \Gamma^{n-1} \times [1..max]^{n-1} \times \prod_{j \neq r} 2^{\mathbb{PS}_{j,>c}}$$

where  $\mathbb{PS}_{j,>c} = \bigcup_{i=c+1}^k \mathbb{PS}_{j,i}$ .

A phase summary  $(q, c, m, \mathcal{T}, \mathcal{M}, \mathcal{S})$  in  $\mathbb{PS}_{r,c}$  still contains the information required to skip a play from a push on stack  $r$  to the matching pop. The matching pop is defined to occur in phase  $c$ . The meaning of  $q, m, \mathcal{T}, \mathcal{M}$  is unchanged from Definition 3. The sets of summaries for stacks  $j \neq r$  refer to phases later than  $c$ . If a symbol is popped in phase  $c$  from stack  $r$  then stack  $j$  can only be popped in a later phase.

When considering context-bounded rather than phase-bounded MPDG, the key insight is that the summaries for the first and the second context can be simplified. A summary for these contexts describes a situation where the push and the matching pop happen within the same context. As a consequence, the other stacks will not change between the push and the pop. This means a summary for context one and two does not need to contain entries for other stacks. This yields the following optimization of Definition 6.

► **Definition 7.** Consider stack  $r$ . We define the set of context summaries by  $\mathbb{CS}_{r,c} = \mathbb{PS}_{r,c}$  for  $c$  from 3 to  $k$ . For  $c = 1, 2$ , the stack has no influence on the definition:

$$\mathbb{CS}_c = Q \times \{c\} \times [1..max] .$$

The largest set of summaries is  $\mathbb{CS}_{r,3}$ , which has size  $\exp_{k-3}(\mathcal{O}(|G|^{2k}))$  or  $\exp_{k-3}(\text{poly}(|G|))$  since  $k$  is fixed, Appendix A.2. Note that the optimization in Definition 7 is not sound for phase-bounded MPDG. There, symbols can be pushed on all stacks in phases one and two. The difference also manifests itself in the lower bound.

### 5 Lower Bounds

We show lower bounds on the complexity of solving context-bounded, phase-bounded, and ordered MPDG. They match the upper bounds established in the previous sections. The lower bounds already hold for reachability as the winning condition and thus carry over to parity. Interestingly, for phase and context-bounded MPDG we only need two stacks.

► **Theorem 8.** Solving  $(k + 2)$ -context respectively  $k$ -phase 2-stack reachability MPDG is  $k$ -EXPTIME-hard. Solving ordered  $n$ -stack reachability MPDG is  $n$ -EXPTIME-hard.

The proofs are by reduction from the membership problem for space-bounded alternating Turing machines [27]. We want to focus on the main ideas. A detailed presentation can be found in Appendix D and [60]. Let  $M$  be an alternating Turing machine that is guaranteed to terminate (decider) and operate with space bound  $\exp_{k-1}(\text{poly}(|w|))$ . Given an input word  $w$ , we show how to construct a 2-stack reachability MPDG  $G_w$  satisfying the following.

► **Lemma 9.** Eve wins  $G_w$  if and only if  $w \in L(M)$ . No play in  $G_w$  exceeds  $(k + 2)$  contexts and  $k$  phases. The construction of  $G_w$  works in time polynomial in  $|w|$ .

Note that the same MPDG  $G_w$  proves the lower bound for the context-bounded and for the phase-bounded case. Appendix D.4 explains how to adapt the construction to ordered MPDG. In that setting, we need  $n$  stacks. Together, this proves Theorem 8.

We give the construction of  $G_w$  in three steps. First, we explain the overall idea of how  $G_w$  simulates  $M$ . Next, we present the key techniques used in the construction, first-order relations defined by a fragment of first-order logic, and Stockmeyer's nested indexing [59, 24, 36]. Finally, we give details of the construction.

## 5.1 Reduction

We recall the semantics of alternating Turing machines. Configurations  $(q, c)$  of  $M$  on input  $w$  consist of a state  $q$  and tape content  $c$ . States are defined to be existentially or universally branching. We generalize this terminology to configurations and speak of existential and universal configurations, respectively. The tape content is a word over the tape alphabet together with a marker denoting the head of the alternating Turing machine. We do not use an additional work tape. A computation of  $M$  on  $w$  yields a tree. The nodes are configurations, the edges are transitions. Existential configurations have one successor configuration, if a transition is possible. Universal configurations have a successor for each possible transition. A configuration is final if it is universal and no transitions are possible. The tree is accepting if every branch reaches a final configuration. There may be different computation trees and  $M$  accepts  $w$  if one of them is accepting.

Configurations  $(q, c)$  of the alternating Turing machine will be modeled by positions in the game  $G_w$ . Alternation will be reflected by the ownership function: Eve will own the positions modeling existential configurations, and hence decide about the transition to take from there. Transitions between configurations will be mimicked by moves. A play of  $G_w$  reflects a branch in some computation tree of  $M$  on input  $w$ . A winning strategy for Eve will yield a computation tree of  $M$  on  $w$  that is accepting. A winning strategy for Ana will find a branch that violates acceptance in any computation tree.

When modeling configuration  $(q, c)$ , state  $q$  will be the control state of  $G_w$ . To understand how tape content is stored, consider a computation branch of  $M$  that leads to  $(q, c)$ . It is a sequence of configurations  $(q_0, c_0) \dots (q_m, c_m)(q, c)$ . The game stores the tape contents on the first stack, in the form  $c\#c_m\#\dots\#c_0$ . The owner of  $q$  chooses a transition  $\delta$  to take from  $(q, c)$ , that results in a configuration  $(q', c')$ . The game pushes the tape content  $c'$  onto the first stack and sets the new control state to  $q'$ .

The difficulty is that tape content  $c'$  is of size  $\exp_{k-1}(\text{len})$  with  $\text{len} = \text{poly}(|w|)$  while the size of  $G_w$  has to remain polynomial. This means the tape content cannot be pushed in a faithful way by only using the control states of the MPDG. Instead, we let Eve propose a sequence of symbols  $\gamma$  from an appropriate alphabet. Afterwards, we give Ana the opportunity to check the sequence for correctness. Correctness is expressed by a number of relations between  $\gamma$  and its predecessor  $c$  on the first stack.

For each relation, we show how to construct a verification mechanism, an MPDG that is entered when Ana chooses to check correctness. Once entered, the verification mechanism cannot be left again. It is constructed in such a way that Eve has a winning strategy from the entry point if and only if the topmost sequences  $\gamma$  and  $c$  on the first stack are in the required relation. The use of verification mechanisms forces a winning strategy for Eve in the overall game to push a sequence  $\gamma$  that satisfies all relations.

Consider the verification mechanism required to implement the relation of a Turing machine transition  $\delta$ , say from configuration  $(q, c)$  to  $(q', c')$ . The verification mechanism has to check that  $\gamma = c'$ , the proposed word is the tape content of the successor configuration. If not, then a single position will witness the mismatch. It is either a position that changed without having the head, or the change did not respect  $\delta$ . The verification mechanism thus needs to compare the same position in  $\gamma$  and  $c$ . Still, the number of positions is not polynomial and verification mechanisms cannot store the position in the control state.

The idea is to annotate the letters in  $c$  and in  $\gamma$  with their positions. This reduces counting to the problem of comparing annotations. However, even if the positions are encoded binary the annotation is still  $\text{exp}_{k-2}(\text{len})$  long. The problem can be addressed in the same way, we again annotate the letters of the encoding with their positions. This recursive process is known as *Stockmeyer's nested indexing* [59, 24, 36], written here as function  $\text{enc}$ . As a consequence, the first stack will actually hold the sequence  $\text{enc}(c)\#\text{enc}(c_m)\#\dots\#\text{enc}(c_0)$ .

The relations between  $\gamma$  and  $\text{enc}(c)$  that Ana will have to check by means of verification mechanisms are: is  $\gamma$  a correct encoding at all,  $\gamma = \text{enc}(u)$  for some  $u$ , and is the encoded tape content  $u$  the successor  $c'$  of content  $c$  according to a Turing machine transition  $\delta$ . We already discussed how to check the latter relation. For the former, the essence is to check the binary increment relation. We rely on further auxiliary relations.

Our key observation is that all relations required for the reduction are defined by a finite alternation of quantifiers over the set of positions. We introduce *first-order relations*, a formalism sufficiently expressive to capture each of these relations. Then we show how to construct a verification mechanism for any first-order relation. This is the main technical contribution of the section.

## 5.2 First-Order Relations

A first-order relation is a relation  $u \sim_\varphi v$  between words  $u$  and  $v$  that is defined by a closed formula  $\varphi$  from a fragment of first-order logic. For the definition of the fragment, let  $\Sigma$  be a finite alphabet ranged over by  $s$ . Let  $\mathcal{V}$  be a countable set of so-called position variables ranged over by  $y$ . A term  $t$  is either an alphabet symbol or the symbol at position  $y$  in the first or in the second word. A formula  $\varphi$  quantifies over positions, compares positions, and compares symbols given by terms  $t_1$  and  $t_2$ :

$$t ::= s \mid \text{symb}_1(y) \mid \text{symb}_2(y) \quad \varphi ::= y_1 \leq y_2 \mid t_1 = t_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists y.\varphi.$$

The remaining logical connectives, the universal quantifier, and common predicates are defined as abbreviations. A formula is closed if every position variable is bound by a quantifier.

Formulas  $\varphi$  are evaluated over pairs of words  $u = u_0 \dots u_{n-1}$  and  $v = v_0 \dots v_{n-1}$  over  $\Sigma$  of the same length together with a valuation of the free position variables  $\text{val} : \mathcal{V} \rightarrow [0..n-1]$ . The semantics of terms is

$$\llbracket s \rrbracket_{u,v}^{\text{val}} = s \quad \llbracket \text{symb}_1(y) \rrbracket_{u,v}^{\text{val}} = u_{\text{val}(y)} \quad \llbracket \text{symb}_2(y) \rrbracket_{u,v}^{\text{val}} = v_{\text{val}(y)}.$$

The semantics of first-order formulas is as expected [30]. For closed formulas, it is independent of the valuation. A closed formula  $\varphi$  defines a so-called *first-order relation* among words that contains all models of the formula,  $\sim_\varphi = \{(u, v) \mid u, v \models \varphi\}$ . To give an example, the ordering  $<$  on natural numbers in most-significant-bit-first encoding is defined by

$$\exists y_1. \forall y_2. [(y_2 < y_1) \rightarrow \text{symb}_1(y_2) = \text{symb}_2(y_2)] \wedge \text{symb}_1(y_1) = 0 \wedge \text{symb}_2(y_1) = 1.$$

A first-order formula is in prenex normal form if it has the shape  $Q_1 y_1 \dots Q_m y_m. \varphi$ , where  $Q_i \in \{\exists, \forall\}$  and  $\varphi$  does not contain quantifiers. Every first-order formula can be transformed into an equivalent formula in prenex normal form [30].

## 5.3 Stockmeyer's Nested Indexing

Our reduction relies on Stockmeyer's nested indexing [59, 24, 36]. It takes a word of length  $\text{exp}_d(n)$  and appends to each letter an index. The index is the letter's position given in most-significant-bit-first encoding. Each index has length  $\text{exp}_{d-1}(n)$ . This encourages to index it as well, and do so on until the indices have polynomial length.

For each nesting depth  $d > 0$  of indices we introduce the alphabet  $\Sigma_d = \{0_d, 1_d\}$ . Let function  $msbf_d$  assign to a number its *most-significant-bit-first encoding* over  $\Sigma_d$ . We define the  $d$ -nested indexing  $ind_d(u)$  of words  $u = u_0 \dots u_m$  by induction. The base case is  $ind_0(u) = u$ , the word itself and the inductive case is

$$ind_{d+1}(u) = u_0 x_0 \dots u_{m-1} x_{m-1}, \quad \text{where } x_i = ind_d(msbf_{d+1}(i)).$$

To give an example,  $ind_2(\text{abra}) = a0_20_10_21_1b0_20_11_21_1r1_20_10_21_1a1_20_11_21_1$ .

In our reduction, all words indexed by  $ind_d$  are of length  $exp_d(len)$  for some  $len \in \mathbb{N}$ . Then all indices in each layer  $d$  have the same length and their  $msbf_d$  encoding ranges from  $0^{exp_{d-1}(len)}$  to  $1^{exp_{d-1}(len)}$ . Since the tape contents in the reduction are of length  $(k-1)$ -fold exponential in the input word of the Turing machine, we define this to be our encoding,  $enc(c) = ind_{k-1}(c)$ . As a result, the lowest layer indices are of polynomial length.

## 5.4 Verification Mechanisms

We proceed by induction on the depth  $d$  of the nested indexing. We show how to construct for every closed first-order formula  $\varphi$  a verification mechanism, a 2-stack MPDG  $G_\varphi^d$  that *decides*  $\sim_\varphi$  over words of length  $exp_d(len)$  in the following sense. We have  $u \sim_\varphi v$  if and only if Eve has a winning strategy from the initial position with  $u, v \in \Sigma^{exp_d(len)}$  on top of the first stack. The initial position takes the shape

$$(\text{Check}_\varphi^d, ind_d(u)\gamma_1 ind_d(v)\gamma_2, \gamma_3),$$

where  $\gamma_1, \gamma_2, \gamma_3$  are arbitrary stack contents up to some delimiting symbols. For the reduction, we want to verify the relation of a Turing machine transition  $\delta$  between encoded configurations of length  $(k-1)$ -fold exponential in the input. We invoke the next lemma with  $d = k-1$  and  $len = \text{poly}(|w|)$  and  $w$  the input to  $M$ . The verification mechanism thus takes at most  $k+1$  contexts and  $k$ , once entered. The first phase for pushing the configurations has no fixed stack, so it merges with the first phase of the verification mechanism for a total of  $k+2$  contexts and  $k$  phases.

► **Lemma 10.** *Let  $\varphi$  be a first-order formula over  $\Sigma$  and  $d \in \mathbb{N}$ . In time  $\text{poly}(d + |\Sigma| + len)$  we can construct a 2-stack reachability MPDG  $G_\varphi^d$  that decides  $\varphi$  over words of length  $exp_d(len)$ . Any play takes at most  $d+2$  contexts and  $d+1$  phases.*

We explain the main ideas behind the construction. Details can be found in Appendix D and [60]. Let  $\varphi = Qy_1 \dots Qy_m.\psi$  be a closed first-order formula in prenex normal form. The game reflects the choice of a valuation  $val$  for  $y_1, \dots, y_m$ , discharging the quantifier alternation to the players. For each  $y_j$ , the responsible player chooses a position  $val(y_j)$  whose binary representation has length  $exp_{d-1}(len)$ . Then she pushes the encoding  $ind_{d-1}(val(y_j))$  on the second stack (cf. Appendix D.3). When all variables have been processed, the second stack holds a sequence  $y_m ind_{d-1}(msbf_d(val(y_m))) \dots y_1 ind_{d-1}(msbf_d(val(y_1)))$  representing  $val$ , where  $y_1, \dots, y_m$  serve as delimiting symbols.

After  $val$  has been chosen, we are interested in the value  $\llbracket \psi \rrbracket_{u,v}^{val}$ . Eve wins if and only if it is true. The difficult case is to evaluate the atomic formulas in  $\psi$ . The idea is to let Eve propose auxiliary information about  $u, v$ , and  $y_1, \dots, y_m$ , which is sufficient to evaluate the atomic formulas. The size of this information is independent from  $d, len$ , so it can be stored in the control state. Together, this means the game does not need to access the stack during evaluation. As before, Ana may verify the proposed information.

Atomic formulas can take the shape  $\llbracket symb_{1/2}(y) \rrbracket_{u,v}^{val}$ . Eve proposes symbol assignments  $symb_u, symb_v : \{y_1, \dots, y_m\} \rightarrow \Sigma$ , mapping each variable  $y$  stored on stack two to the symbol of word  $u, v$  at the position  $val(y)$ . If Ana chooses to verify  $symb_u(y)$  (resp.  $symb_v(y)$ ) for some  $y$ , Eve removes symbols from stack one until she claims to have found position  $val(y)$ . Ana then decides to either compare the symbol found on stack one to  $symb_u(y)$  ( $symb_v(y)$ ) or verify the equality of the valuation  $ind_{d-1}(val(y))$  on stack two and the annotated index on stack one (Appendix D.1).

Alternatively, an atomic formula can be  $\llbracket y \leq y' \rrbracket_{u,v}^{val}$ . For these, Eve proposes a variable order, a sequence of variables interleaved with relations of the form

$$y_{l_1} \theta_1 y_{l_2} \theta_2 \dots \theta_{m-1} y_{l_m}, \quad \text{with } \theta_j \in \{=, <, >\}.$$

Verifying an entry  $y \theta y'$  amounts to verifying one of the first-order relations  $=$ ,  $<$ , and  $>$  on the corresponding valuations stored on stack two. This is an application of induction, since the valuations are of smaller nested depth, i.e.  $ind_{d-1}(val(y))$  and  $ind_{d-1}(val(y'))$ .

Note that the number of possible  $symb_u, symb_v$ , and variable orders is independent of the indexing depth  $d$  and the input length  $len$ , but exponential in the size of  $\varphi$ .

**Contexts and Phases.** The verification mechanism begins by pushing the valuation onto stack two. This process either succeeds with the final valuation on stack two, which costs one context or phase, but does not fix the stack for the phase, or fails and takes at most  $d + 1$  contexts and  $d$  phases (Appendices D.3, D.2).

Then, Eve proposes information. Ana may doubt that  $symb_1(y)$  or  $symb_2(y)$  equals  $u_{val(y)}$  resp.  $v_{val(y)}$ . Popping the first stack introduces a second context and sets the stack for the first phase. The comparison routine adds  $(d - 1) + 2$  contexts and phases (Appendix D.1). Since the context and phase for popping merges with the first context or phase of this routine, the resulting play has up to  $d + 2$  contexts and  $d + 1$  phases.

Alternatively, Ana may doubt an entry  $y \theta y'$  of the variable order. This includes removing irrelevant variable values from stack two, which continues the first context and sets the stack for the first phase. Then  $C_{\varphi_\theta}^{d-1}$  adds up to  $(d - 1) + 2$  contexts and  $(d - 1) + 1$  phases by the induction hypothesis. This yields a bound of  $d + 2$  contexts and  $d$  phases.

If Ana believes the proposal, the game ends without a further context or phase. The maximum across all plays is thus  $d + 2$  contexts and  $d + 1$  phases.

---

## References

- 1 CPACHECKER. URL: <https://cpachecker.sosy-lab.org/index.php>.
- 2 ULTIMATE AUTOMIZER and ULTIMATE TAIPAN. URL: <https://monteverdi.informatik.uni-freiburg.de/tomcat/Website/>.
- 3 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *LMCS*, 3(3), 2007.
- 4 S. Akshay, P. Gastin, S. Krishna, and S. Roychowdhury. Revisiting underapproximate reachability for multipushdown systems. In *TACAS*, volume 12078 of *LNCS*, pages 387–404. Springer, 2020. doi:10.1007/978-3-030-45190-5\_21.
- 5 M. F. Atig. From multi to single stack automata. In *CONCUR*, volume 6269 of *LNCS*, pages 117–131. Springer, 2010.
- 6 M. F. Atig. Global model checking of ordered multi-pushdown systems. In *FSTTCS*, volume 8 of *LIPICs*, pages 216–227. Dagstuhl, 2010.
- 7 M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. Model checking branching-time properties of multi-pushdown systems is hard. *CoRR*, abs/1205.6928, 2012. URL: <http://arxiv.org/abs/1205.6928>.

- 8 M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. Parity games on bounded phase multi-pushdown systems. In *NETYS*, volume 10299 of *LNCS*, pages 272–287. Springer, 2017.
- 9 T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL*, pages 221–234. ACM, 2014.
- 10 T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. Recursive games for compositional program synthesis. In *VSTTE*, volume 9593 of *LNCS*, pages 19–39. Springer, 2015.
- 11 D. Beyer. Advances in automatic software verification: SV-COMP 2020. In *TACAS*, volume 12079 of *LNCS*, pages 347–367. Springer, 2020.
- 12 D. Beyer, M. Dangl, and P. Wendler. A unifying view on SMT-based software verification. *JAR*, 60(3):299–335, 2018.
- 13 D. Beyer and M. E. Keremoglu. CPACHECKER: A tool for configurable software verification. In *Proc. CAV*, volume 6806 of *LNCS*, pages 184–190. Springer, 2011.
- 14 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
- 15 A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *FSTTCS*, volume 3328 of *LNCS*, pages 135–147. Springer, 2004.
- 16 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
- 17 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP*, volume 7392 of *LNCS*, pages 165–176. Springer, 2012.
- 18 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-SHORE: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24. ACM, 2013.
- 19 C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, volume 23 of *LIPICs*, pages 129–148. Dagstuhl, 2013.
- 20 J. R. Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6(3):91–111, 1964.
- 21 T. Cathcart Burn, C.-H. L. Ong, and S. J. Ramsay. Higher-order constrained Horn clauses for verification. *Proc. ACM Program. Lang.*, 2(POPL):11:1–11:28, 2018.
- 22 Aiswarya C. *Verification of communicating recursive programs via split-width*. PhD thesis, ENS Cachan, 2014.
- 23 T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.
- 24 T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007. URL: <http://arxiv.org/abs/0705.0262>.
- 25 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *AFL*, volume 151 of *EPTCS*, pages 1–24, 2014.
- 26 A. Carayol, M. Hague, A. Meyer, C.-H. L. Ong, and O. Serre. Winning regions of higher-order pushdown games. In *LICS*, pages 193–204. IEEE, 2008.
- 27 A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *JACM*, 28(1):114–133, 1981.
- 28 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic*. CUP, 2012.
- 29 A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012. doi: 10.1007/978-3-642-32940-1\_38.
- 30 H. B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.
- 31 J. Esparza. On the decidability of model checking for several  $\mu$ -calculi and Petri nets. In *CAAP*, volume 787 of *LNCS*, pages 115–129. Springer, 1994.
- 32 A. Finkel, B. Willem, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Infinity*, volume 9 of *ENTCS*, pages 27–37. Elsevier, 1997.
- 33 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

- 34 C. Grellois and P.-A. Mellès. Finitary semantics of linear logic and higher-order model-checking. In *MFCS*, volume 9234 of *LNCS*, pages 256–268. Springer, 2015.
- 35 M. Hague. Saturation of concurrent collapsible pushdown systems. In *FSTTCS*, volume 24 of *LIPICs*, pages 313–325. Dagstuhl, 2013.
- 36 M. Hague, R. Meyer, and S. Muskalla. Domains for Higher-Order Games. In *MFCS*, volume 83 of *LIPICs*, pages 59:1–59:15. Dagstuhl, 2017.
- 37 M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *LMCS*, 4(4), 2008.
- 38 M. Heizmann, Y.-W. Chen, D. Dietsch, M. Greitschus, A. Nutz, B. Musa, C. Schätzle, C. Schilling, F. Schüssele, and Andreas Podelski. ULTIMATE AUTOMIZER with an on-demand construction of Floyd-Hoare automata. In *TACAS*, volume 10206 of *LNCS*, pages 394–398. Springer, 2017.
- 39 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
- 40 T. A. Henzinger. Games in system design and verification. In *TARK*, pages 1–4. National University of Singapore, 2005.
- 41 M. Hofmann and J. Ledent. A cartesian-closed category for higher-order model checking. In *LICS*, pages 1–12. IEEE, 2017.
- 42 M. Jurdzinski. Small progress measures for solving parity games. In *STACS*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3.
- 43 M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *LICS*, pages 1–9. IEEE, 2017.
- 44 M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. C.*, 38(4):1519–1532, 2008.
- 45 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, pages 416–428. ACM, 2009.
- 46 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, pages 179–188. IEEE, 2009.
- 47 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE, 2007.
- 48 S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.
- 49 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.
- 50 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294. ACM, 2011.
- 51 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE, 2006.
- 52 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
- 53 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM ToPLaS*, 22(2):416–430, 2000.
- 54 T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, pages 49–61. ACM, 1995.
- 55 S. Salvati and I. Walukiewicz. A model for behavioural properties of higher-order programs. In *CSL*, volume 41 of *LIPICs*, pages 229–243. Dagstuhl, 2015.
- 56 A. Seth. Games on higher order multi-stack pushdown systems. In *RP*, volume 5797 of *LNCS*, pages 203–216. Springer, 2009.
- 57 A. Seth. Games on multi-stack pushdown systems. In *LFCS*, volume 5407 of *LNCS*, pages 395–408. Springer, 2009.

- 58 M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. TR 2, NYU, 1978.
- 59 L. J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, MIT, 1974.
- 60 S. van der Wall. Bounded analysis of concurrent and recursive programs. Master's thesis, TU Braunschweig, 2019. URL: <http://www.tcs.cs.tu-bs.de/documents/thesis-van-der-wall-2019.pdf>.
- 61 I. Walukiewicz. Pushdown processes: games and model-checking. *IC*, 164(2):234–263, 2001.
- 62 I. Walukiewicz. A landscape with games in the background. In *LICS*, pages 356–366. IEEE, 2004.

## A Details on Section 3

We argue that Theorem 1 follows from Lemma 2. To check whether Eve wins  $G$ , by the first statement in the lemma it is sufficient to check whether she wins  $F$ . We thus construct the finite game  $F$ , which takes  $(k - 2)$ -fold exponential time. We apply a modern parity game solving algorithm to determine the winner in  $F$ , like the recent subexponential algorithms [43, 44]. The algorithm takes time exponential only in the priorities of  $F$ , which by the second statement in the lemma are the priorities  $[0..max]$  in  $G$ . We thus obtain an overall time complexity

$$\exp_{k-2}(\text{poly}(|G|)) + \exp_{k-2}(\text{poly}(|G|))^{max} \leq \exp_{k-2}((1 + max) \text{poly}(|G|)),$$

which is still  $\exp_{k-2}(\text{poly}(|G|))$ .

### A.1 Size of the sets of ordered summaries

We estimate the size of the optimized sets of summaries by induction on the stack. In the base case:

$$|\mathbb{OS}_n| = |Q| \cdot max = \exp_{n-n}(\mathcal{O}(|G|^2)).$$

For the induction step, assume  $|\mathbb{OS}_{r+1}| = \exp_{n-(r+1)}(\mathcal{O}(|G|^2))$ . For stack  $r$  we obtain

$$\begin{aligned} |\mathbb{OS}_r| &= |Q| \cdot \prod_{j=r+1}^n |\Gamma| \cdot max \cdot 2^{\sum_{i=r+1}^n |\mathbb{OS}_i|} \leq |Q| \cdot |\Gamma|^{n-r} \cdot max^{n-r} \cdot \prod_{j=r+1}^n 2^{n|\mathbb{OS}_{r+1}|} \\ &\leq |G|^{2(n-r)+1} \cdot 2^{(n-r)n|\mathbb{OS}_{r+1}|} \leq 2^{(2n+1)|G|+n^2|\mathbb{OS}_{r+1}|} \end{aligned}$$

The equality is by definition. The first inequality relies on  $|\mathbb{OS}_r| \geq \dots \geq |\mathbb{OS}_n|$ . Since  $n$  is a constant,

$$2^{(2n+1)|G|+n^2|\mathbb{OS}_{r+1}|} = 2^{(2n+1)|G|+n^2 \exp_{n-(r+1)}(\mathcal{O}(|G|^2))} = \exp_{n-r}(\mathcal{O}(|G|^2)).$$

### A.2 Size of the sets of context summaries

We estimate the size of the optimized sets of summaries by induction on the context. In the base case:

$$|\mathbb{OS}_{r,1}| = |\mathbb{OS}_{r,2}| \leq |\mathbb{OS}_{r,k}| \leq |Q| \cdot max \cdot |\Gamma|^{k-1} \cdot max^{k-1} = \exp_{k-k}(\mathcal{O}(|G|^{2k})).$$

## 52:18 On the Complexity of Multi-Pushdown Games

The function is indeed polynomial as the number of stacks  $k$  is fixed. For the induction step, assume  $|\mathbb{OS}_{r,c+1}| = \text{exp}_{k-(c+1)}(\mathcal{O}(|G|^{2k}))$  with  $4 \leq c+1 \leq k$ . For context  $c$  we obtain

$$\begin{aligned} |\mathbb{OS}_{r,c}| &= |Q| \cdot \max \cdot \prod_{j=1}^{r-1} |\Gamma| \cdot \max \cdot 2^{\sum_{l=c+1}^k |\mathbb{OS}_{j,l}|} \\ &\quad \cdot \prod_{j=r+1}^k |\Gamma| \cdot \max \cdot 2^{\sum_{l=c+1}^k |\mathbb{OS}_{j,l}|} \\ &\leq |G|^{2k} \cdot \prod_{j=1}^k 2^{k|\mathbb{OS}_{j,c+1}|} \leq 2^{2k|G|+k^2|\mathbb{OS}_{r,c+1}|}. \end{aligned}$$

The equality is by definition. The first inequality relies on  $|\mathbb{OS}_{r,c}| \geq \dots \geq |\mathbb{OS}_{r,k}|$ . Also,  $|\mathbb{OS}_{j,c}| = |\mathbb{OS}_{j',c}|$  for stacks  $j \neq j'$  by Symmetry.

$$2^{2k|G|+k^2|\mathbb{OS}_{r,c+1}|} = 2^{2k|G|+k^2 \text{exp}_{k-(c+1)}(\mathcal{O}(|G|^{2k}))} = \text{exp}_{k-c}(\mathcal{O}(|G|^{2k})).$$

### **B** Equivalence of the MPDG $G$ and the finite game $F$

In the following, we give a construction intuition for how winning strategies for Eve can be converted between  $G$  and  $F$ .

#### **B.1** Transforming a winning strategy from $F$ to $G$

We proceed in the following steps: First, we introduce a strategy transducer  $T$  to transport  $\sigma$  from  $F = (V_F, E_F, \text{own}, \Omega)$  to  $G = (P, \text{own}, \Omega)$ . Then, we define the strategy  $\nu$  it implements and show an invariant between plays compliant with that strategy and runs of  $T$  (Lemma 14). Next, we show that if  $T$  can perform a run from a stair  $(q, \mathcal{R})$  to some configuration  $(q', \mathcal{R}')$ , then there is a play compliant to  $\sigma$  in  $F$  from  $\text{Tr}(q, \mathcal{R})$  to  $\text{Tr}(q', \mathcal{R}')$  (Lemma 15 and Lemma 16). Lastly, putting the previous results together, we get that a run compliant to  $\nu$  that is losing for Eve leads to a play in  $F$  that is compliant to  $\sigma$  losing for Eve, which contradicts it being a winning strategy. Thus,  $\nu$  is also a winning strategy.

Intuitively, the transducer  $T$  is a multi-pushdown system with the same number of stacks as  $P$ . At any point in the game, the stack heights of  $T$  are identical to the stack heights of  $P$ . However, the transitions are amplified to update all top of stack contents with each transition.

The strategy automaton remembers information of the finite state game in the following sense. If the finite state game would be in a position  $(\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$ , then the strategy automaton is in state  $q$  and the top of stack symbol of each stack  $r$  is a tuple  $(\mathcal{T}[r], \mathcal{M}[r], \mathcal{S}[r])$ . The automaton mimics a play of  $G$ . Whenever Eve has the next move, she can use it to follow her strategy  $\sigma$  for  $F$ .

► **Definition 11.** *Given a strategy  $\sigma$  for Eve in  $F$ , the strategy automaton  $T$  is a tuple  $T = (Q, \Gamma \times [0..max] \times 2^{\mathbb{OS}}, \mapsto, n)$  where  $Q$  is the state space,  $\Gamma \times [0..max] \times 2^{\mathbb{OS}}$  is the stack alphabet, and  $\mapsto$  is a transition relation, which we will define directly on the set of configurations  $C_T$ .*

A configuration of  $T \in C_T$  is  $(q, \mathcal{R})$ , where  $\mathcal{R} : [1..n] \rightarrow (\Gamma \times [0..max])^* \times 2^{\mathbb{OS}}$  are the stack contents. For each stack  $j$ , the stack contents  $\mathcal{R}[j] = {}^j\mathcal{R}_{|\mathcal{R}[j]|} {}^j\mathcal{R}_{|\mathcal{R}[j]|-1} \dots {}^j\mathcal{R}_1$  is a sequence of tuples, and  ${}^j\mathcal{R}_{|\mathcal{R}[j]|}$  is the top of stack symbol. We denote the tuple contents by  ${}^j\mathcal{R}_i = ({}^j\gamma_i, {}^j m_i, {}^j S_i)$ .

For the sake of notation, we introduce a context sensitive top of stack pointer  $\uparrow$ . It obtains the index value of the top of stack symbol:

$$\mathcal{R}[j] = {}^j\mathcal{R}_{|\mathcal{R}[j]|} {}^j\mathcal{R}_{|\mathcal{R}[j]|\!-\!1} \dots {}^j\mathcal{R}_1 = {}^j\mathcal{R}_{\uparrow} {}^j\mathcal{R}_{\uparrow\!-\!1} \dots {}^j\mathcal{R}_1.$$

This notation also carries over to the individual tuple contents. Thus,

$${}^j\gamma_{\uparrow} = {}^j\gamma_{|\mathcal{R}[j]|} \quad {}^jS_{\uparrow} = {}^jS_{|\mathcal{R}[j]|} \quad {}^jm_{\uparrow} = {}^jm_{|\mathcal{R}[j]|}.$$

► **Definition 12.** We define a transformation function  $Tr : C_T \rightarrow V_F$  mapping configurations of  $T$  to positions in  $F$  by  $Tr(q, \mathcal{R}) = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$ , where  $\mathcal{T}[j] = {}^j\gamma_{\uparrow}$ ,  $\mathcal{M}[j] = {}^jm_{\uparrow}$  and  $\mathcal{S}[j] = {}^jS_{\uparrow}$ .

To define the transition relation  $\mapsto \subseteq C_T \times \delta \times C_T$ , let  $Tr(q, \mathcal{R}) = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$  and  $Tr(q', \mathcal{R}') = (\text{Check}, q', \mathcal{T}', \mathcal{M}', \mathcal{S}')$ . For every transition rule  $\tau$  of  $P$ ,  $T$  has a transition  $(q, \mathcal{R}) \xrightarrow{\tau} (q', \mathcal{R}')$ , if either  $q$  is owned by Eve and  $\sigma$  tells her for position  $Tr(q, \mathcal{R})$  to use the move simulating  $\tau$ , or  $own(q) = \text{Ana}$ .

Further, for all stacks  $j \neq r$ ,  $\mathcal{T}'[j] = {}^j\gamma'_{\uparrow} = {}^j\gamma_{\uparrow} = \mathcal{T}'[j]$  and  $T$  updates the stackcontents  $\mathcal{R}$  to  $\mathcal{R}'$  dependent on the transition:

**Case 1** ( $\tau$  is an internal transition  $(q, r, q')$ ):  $\mathcal{R} = \mathcal{R}'$  except for each stack  $j$ ,  ${}^jm'_{\uparrow} = \max\{\Omega(q'), {}^jm_{\uparrow}\}$ .

**Case 2** ( $\tau$  is a push transition  $(q, r, s, q')$ ):  $\mathcal{R} = \mathcal{R}'$ , except for each stack  $j \neq r$ ,  ${}^jm'_{\uparrow} = \max\{\Omega(q'), {}^jm_{\uparrow}\}$ . And for stack  $r$ ,  $\mathcal{R}'[r] = (s, \Omega(q'), \mathcal{S})\mathcal{R}[r]$ , where  $\mathcal{S}$  is determined by  $\sigma$ :

$$\sigma(\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s) = (\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s, \mathcal{S}).$$

**Case 3** ( $\tau$  is a pop transition  $(q, s, r, q') \in \delta_{pop}$  and  $\mathcal{R}[j] = \varepsilon$  for all  $j < r$ ):

*Case 3.1* (there is  $(q', \mathcal{M}[r], \mathcal{T}_{[r \rightarrow \varepsilon]}, \mathcal{M}_{[r \rightarrow 0]}, \mathcal{S}) \in {}^rS_{\uparrow}$  s.t. for each  $j > r$ ,  $\mathcal{S}[j] \subseteq {}^jS_{\uparrow}$ ):  $\mathcal{R} = \mathcal{R}'$ , except for each  $j > r$ ,  ${}^jm'_{\uparrow} = \max\{\Omega(q'), {}^jm_{\uparrow}\}$  and  ${}^jS'_{\uparrow} = \mathcal{S}[j]$ . And for stack  $r$ ,  $\mathcal{R}'[r] = ({}^r\gamma_{\uparrow\!-\!1}, \max\{{}^rm_{\uparrow}, {}^rm_{\uparrow\!-\!1}, \Omega(q')\}, {}^rS_{\uparrow\!-\!1}){}^r\mathcal{R}_{\uparrow\!-\!2}{}^r\mathcal{R}_{\uparrow\!-\!3} \dots {}^r\mathcal{R}_1$ .

*Case 3.2* (there is **no**  $(q', \mathcal{M}[r], \mathcal{T}_{[r \rightarrow \varepsilon]}, \mathcal{M}_{[r \rightarrow 0]}, \mathcal{S}) \in {}^rS_{\uparrow}$  s.t. for each  $j > r$ ,  $\mathcal{S}[j] \subseteq {}^jS_{\uparrow}$ ):  $\mathcal{R} = \mathcal{R}'$ , except for each  $j > r$ ,  ${}^jm'_{\uparrow} = \max\{\Omega(q'), {}^jm_{\uparrow}\}$  and for stack  $r$ ,  $\mathcal{R}'[r] = ({}^r\gamma_{\uparrow\!-\!1}, \max\{{}^rm_{\uparrow}, {}^rm_{\uparrow\!-\!1}, \Omega(q')\}, {}^rS_{\uparrow\!-\!1}){}^r\mathcal{R}_{\uparrow\!-\!2}{}^r\mathcal{R}_{\uparrow\!-\!3} \dots {}^r\mathcal{R}_1$ .

Note that case 3.2 is almost a copy of case 3.1. It simply does not find a matching summary. We will later use case distinction on whether a transition is due to case 3.1 or 3.2. Also note that when a configuration  $(q, \mathcal{R})$  belongs to Eve,  $T$  can only perform the transition which  $\sigma$  wants to simulate from  $Tr(q, \mathcal{R})$ .

The following lemma tells us, that during a run of  $T$ , the summary for a stack symbol can only shrink during the run.

► **Lemma 13.** Let  $\eta = (q, \mathcal{R})$  and  $\eta' = (q', \mathcal{R}')$  with  $\eta \xrightarrow{\tau} \eta'$  in  $T$ . For each stack  $j \in [1..n]$ , let  $sh_j = \min\{|\mathcal{R}[j]|, |\mathcal{R}'[j]|\}$ .

For each stack  $j \in [1..n]$ ,  ${}^jS'_{sh_j} \subseteq {}^jS_{sh_j}$  and for each  $u \in [1..sh_j - 1]$ ,  ${}^j\mathcal{R}_u = {}^j\mathcal{R}'_u$ .

**Proof.** By construction. Only transition case 3.1 changes  ${}^jS_{sh_j}$ . When transition case 3.1 happens for a stack  $r$ , it changes on stacks  $j > r$  the set  ${}^jS_{\uparrow}$  to  $\mathcal{S}[j]$ . But  $\mathcal{S}[j] \subseteq {}^jS_{\uparrow}$  by the conditions for transition 3. ◀

## 52:20 On the Complexity of Multi-Pushdown Games

A run of the strategy automaton is a sequence  $(q_{\text{init}}, (\varepsilon, \emptyset, \Omega(q_{\text{init}})^n) = \eta_0 \xrightarrow{\tau_0} \eta_1 \xrightarrow{\tau_1} \dots$ . We can use the strategy automaton to define a strategy  $\nu$  on  $G$  for starting positions of the form  $(q_{\text{init}}, \varepsilon^n)$ . We define  $\nu$  inductively on the play prefix. The proof of the next lemma does both, it defines the strategy and states an invariant between plays in  $G$  conform to it and runs of  $T$ .

► **Lemma 14.** *Let  $\pi = \pi_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{l-1}} \pi_l$  be a play prefix compliant to the strategy  $\nu$  and  $\eta = \eta_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{l-1}} \eta_l$  the corresponding run of  $T$ . At any position  $p \in \mathbb{N}$ , if  $\eta_p = (q, \mathcal{R})$ , then*

1.  $\pi_p = (q, ({}^1\gamma_{\uparrow} {}^1\gamma_{\uparrow-1} \dots {}^1\gamma_1, {}^2\gamma_{\uparrow} {}^2\gamma_{\uparrow-1} \dots {}^2\gamma_1, \dots, {}^n\gamma_{\uparrow} {}^n\gamma_{\uparrow-1} \dots {}^n\gamma_1))$ .
2. *If  $\text{lup}_j^\pi(p) \neq \perp$  is defined, then  $\max_{u \in [\text{lup}_j^\pi(p)..p]} \{\Omega(q^u)\} = {}^j m_{\uparrow}$ . And if  $\text{lup}_j^\pi(p) = \perp$  is undefined, then  $\max_{u \in [0..p]} \{\Omega(q^u)\} = {}^j m_{\uparrow}$ .*

**Proof.**

► **Base Case**  $(\pi_0, \eta_0)$ .  $\pi_0 = (q_{\text{init}}, \varepsilon^n)$ ,  $\eta_0 = ((q_{\text{init}}, 0, 0), (\varepsilon, \emptyset, \Omega(q_{\text{init}})^n))$ . Both invariants hold immediately.

► **Inductive Case**  $(\pi_i \text{ to } \pi_{i+1}, \eta_i \text{ to } \eta_{i+1})$ . Let  $\pi = \pi_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{i-1}} \pi_i = (q, \mathcal{P}) \xrightarrow{\tau_i} (q, \mathcal{P}')$  be a play prefix in  $G$ . By induction, the strategy automaton has a run prefix  $\eta_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{i-1}} \eta_i = (q, \mathcal{R})$ , that fulfills the lemma.

In case of  $\text{own}(\eta_i) = \text{Eve}$ , by construction,  $T$  has only an enabled transition for a single  $\tau$  at  $\eta_i$ . Namely the one, which  $\sigma$  would choose to simulate from  $\psi(q, \mathcal{P})$ . We choose  $\nu(\pi_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{i-1}} \pi_i)$  to use that transition.

In the other case, for every transition  $\tau_i$  enabled in  $\pi_i$ , a corresponding is enabled in  $T$  by construction.

If  $\pi$  is compliant with  $\nu$ , we continue  $\eta$  by the corresponding enabled transition  $\eta_i = (q, \mathcal{R}) \xrightarrow{\tau_i} (q', \mathcal{R}') = \eta_{i+1}$ .

Remains to show, that the lemma holds for position  $i + 1$  as well.

For all stacks  $j \neq r$ , the third condition is fulfilled by construction and induction: By induction, the maximal parity seen since position  $\text{lup}_j^\pi(i)$  up to  $\pi_i$  is  ${}^j m_{\uparrow}$ . For all stacks  $j \neq r$ , the stack height does not change. There has been seen a new parity  $\Omega(q')$ . The construction sets  ${}^j m'_{\uparrow}$  appropriately to  $\max\{\Omega(q'), {}^j m_{\uparrow}\}$ .

**Case 1**  $(\tau_i = (q, r, q') \in \delta_{\text{int}})$ : The first condition is fulfilled trivially, since the symbols in the stack contents did not change. For the second condition, in this case, the same arguments apply as for the other stacks.

**Case 2**  $(\tau_i = (q, r, s, q') \in \delta_{\text{push}})$ : For the first condition, the symbols in the stack contents don't change for all stacks  $j \neq r$ . For stack  $r$  however,  $\mathcal{R}'[r] = (s, \Omega(q'), \mathcal{S})\mathcal{R}[r]$ , such that  $s$  is the new additional symbol, which is the pushed symbol by  $\tau_i$ . This meets the lemma's requirement.

For the third condition,  $\Omega(q')$  is the only parity seen since the push of  $s$ .

**Case 3**  $(\tau_i = (q, s, r, q') \in \delta_{\text{pop}})$ : Be aware that  $T$  contains multiple possible transitions for  $\tau_i$ . These only differ in the sets of summaries  ${}^j \mathcal{S}_{\uparrow}$  for each stack  $j > r$ . The lemma does not state any conditions on the summary sets, so there is no need for case distinction.

For the first condition, the symbols in the stack contents don't change for stacks  $j \neq r$ . For stack  $r$  however,  $(s, \Omega(q'), \mathcal{S})\mathcal{R}[r]' = \mathcal{R}[r]$ , removing the top most symbol from the stack, which is the symbol removed by  $\tau_i$ . This meets the lemma's requirement.

For the second condition, the maximal parity seen since the push of  ${}^r\gamma'_\uparrow$  is determined by the maximal parity seen since the last unmatched push before pushing  $s$ , the just popped symbol. This is the position  $\text{lup}_r^\pi(i+1) = \text{lup}_r^\pi(\text{lup}_r^\pi i)$ .

The correct priority is chosen by  ${}^r m'_\uparrow = \max\{{}^r m_\uparrow, {}^r m_{\uparrow-1}, \Omega(q')\}$ . ◀

The next lemma ensures that the strategy automaton  $T$  and the mapping  $Tr$  of configurations from  $T$  to positions in  $F$  behave well with respect to the strategy  $\sigma$  itself. When the strategy automaton is able to make a move  $\eta \xrightarrow{\tau} \eta'$ , then there should be transitions  $Tr(\eta) \mapsto \dots \mapsto Tr(\eta')$  compliant with  $\sigma$  in  $F$ .

► **Lemma 15.** *Let  $\sigma$  be a strategy for Eve in  $F$  and  $T$  the strategy automaton. Let  $\eta = \eta_0 \xrightarrow{\tau_0} \eta_1 \xrightarrow{\tau_1} \dots$  be a computation of  $T$  starting in a stair  $\eta_0$ .*

*For each position  $i \in \mathbb{N}$ , let*

$$\eta_i = (q^i, \mathcal{R}^i) \quad \text{Tr}(\eta_i) = (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i)$$

where for each stack  $j$ ,  $\mathcal{R}^i[j] = ({}^j\gamma_\uparrow^i, {}^j m_\uparrow^i, {}^j S_\uparrow^i)({}^j\gamma_{\uparrow-1}^i, {}^j m_{\uparrow-1}^i, {}^j S_{\uparrow-1}^i) \dots ({}^j\gamma_1^i, {}^j m_1^i, {}^j S_1^i)$ .

Let  $i \in \mathbb{N}$ . Let  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  be a transition of  $T$  that is not by transition case 3.2.

Then the following transitions exist in  $F$  and are compliant with  $\sigma$ .

If  $\tau_i = (q^i, r, q^{i+1}) \in \delta_{int}$ :

$$\text{Tr}(\eta_i) = (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \mapsto (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = \text{Tr}(\eta_{i+1})$$

$\tau_i = (q, r, s, q') \in \delta_{push}$ :

$$\begin{aligned} \text{Tr}(\eta_i) &= (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \\ &\mapsto (\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s, {}^r S_\uparrow^{i+1}) \\ &\mapsto (\text{Check}, q^{i+1}, \mathcal{T}_{[r \mapsto s]}^i, \mathcal{M}^{i+1}, \mathcal{S}_{[r \mapsto {}^r S_\uparrow^{i+1}]}^i) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

If  $\tau_i = (q, s, r, q') \in \delta_{pop}$ : Since  $\eta_0$  is a stair, position  $i$  is in a push-pop-pair  $(t, i)$ .

$$\begin{aligned} \text{Tr}(\eta_t) &= (\text{Check}, q^t, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t) \\ &\mapsto (\text{Push}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r S_\uparrow^{t+1}) \\ &\mapsto (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r]) \\ &\mapsto (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}, ) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

**Proof.** For any of the following, the correctness of  $q^{i+1}$  is immediate and therefore skipped. Also be aware that both,  $F$  and  $T$ , respect the orderedness restriction.

**Case 1** ( $\tau_i = (q^i, r, q^{i+1}) \in \delta_{int}$ ): By construction of  $F$ ,  $\tau_i$  causes the existence of the transition

$$\begin{aligned} \text{Tr}(\eta_i) &= (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \mapsto (\text{Check}, q^{i+1}, \mathcal{T}^i, \mathcal{M}', \mathcal{S}^i) \\ &\stackrel{!}{=} (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

Remains to show the equality of the last two check states. By definition of  $T$ ,  $\mathcal{R}^i = \mathcal{R}^{i+1}$ . Together with  $Tr$  we get that  $\mathcal{T}^i = \mathcal{T}^{i+1}$  and  $\mathcal{S}^i = \mathcal{S}^{i+1}$ . By construction of  $F$  and  $T$ , for any stack  $j$ ,

$$\mathcal{M}[j]^{i+1} = {}^j m_\uparrow^{i+1} = \max\{{}^j m_\uparrow^i, \Omega(q^{i+1})\} = \max\{\mathcal{M}[j]^i, \Omega(q^{i+1})\} = \mathcal{M}[j]^i.$$

## 52:22 On the Complexity of Multi-Pushdown Games

Also, by construction of  $T$ , the transition  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  only exists, if  $\text{own}(q^i) = \text{Ana}$  or  $\sigma(\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) = (\text{Check}, q^{i+1}, \mathcal{T}^i, \mathcal{M}', \mathcal{S}^i)$ . The transition is compliant with  $\sigma$ .

**Case 2** ( $\tau_i = (q^i, r, s, q^{i+1}) \in \delta_{\text{push}}$ ): By construction of  $F$ ,  $\tau_i$  causes the existence of the transitions

$$\begin{aligned} \text{Tr}(\eta_i) &= (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \\ &\mapsto (\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s, {}^r S_{\uparrow}^{i+1}) \\ &\mapsto (\text{Check}, q^{i+1}, \mathcal{T}_{[r \rightarrow s]}^i, \mathcal{M}', \mathcal{S}_{[r \rightarrow s]}^i) \\ &\stackrel{!}{=} (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

Remains to show the equality of the last two check states. By definition of  $T$ ,  $\mathcal{R}^i = \mathcal{R}^{i+1}$ , except for each  $j \neq r$ ,  ${}^j m_{\uparrow}^{i+1} = \max\{\Omega(q^{i+1}), {}^j m_{\uparrow}^i\}$ , and for stack  $r$ ,  $\mathcal{R}^{i+1}[r] = (s, \Omega(q^{i+1}), {}^r S_{\uparrow}^{i+1})\mathcal{R}[r]$ , where

$$\sigma(\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s) = (\text{Claim}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s, {}^r S_{\uparrow}^{i+1}).$$

This immediately yields  $\mathcal{T}_{[r \rightarrow s]}^i = \mathcal{T}^{i+1}$  and  $\mathcal{S}_{[r \rightarrow s]}^i = \mathcal{S}^{i+1}$ .

Also,  $\mathcal{M}'[r] = \Omega(q^{i+1}) = {}^r m_{\uparrow}^{i+1}$  and for each stack  $j \neq r$ :

$$\mathcal{M}'[j] = \max\{\Omega(q^{i+1}), \mathcal{M}[j]^i\} = \max\{\Omega(q^{i+1}), {}^j m_{\uparrow}^i\} = {}^j m_{\uparrow}^{i+1}.$$

Since the transition  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  exists in  $T$ , by its construction,

$$\sigma(\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s) = (\text{Claim}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s, {}^r S_{\uparrow}^{i+1}).$$

Also, either  $\text{own}(q^i) = \text{Ana}$  or

$$\sigma(\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) = (\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s).$$

Thus, the transitions exist in  $F$  and are compliant with  $\sigma$ .

**Case 3** ( $\tau_i = (q, r, s, q') \in \delta_{\text{pop}}$  and  $(t, i)$  is a push-pop-pair): Since  $(t, i)$  is a push-pop-pair, there is a push transition  $\tau_t = (q^t, r, s, q^{t+1})$ .

By construction of  $F$ ,  $\tau_t$  causes the existence of the transitions

$$\begin{aligned} \text{Tr}(\eta_t) &= (\text{Check}, q^t, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t) \\ &\mapsto (\text{Push}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r S_{\uparrow}^{t+1}) \end{aligned}$$

which are compliant with  $\sigma$ , as discussed in the previous case. Since  $(t, i)$  is a push-pop-pair, for any position  $t < p < i$ ,  $|\mathcal{R}^{t+1}[r]| = |\mathcal{R}^i[r]| \leq |\mathcal{R}^p[r]|$ . Then, by repetitive use of Lemma 13,  ${}^r S_{\uparrow}^i \subseteq {}^r S_{\uparrow}^{t+1}$ .

Next, we show that

$$(\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r S_{\uparrow}^{t+1}) \mapsto (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \rightarrow 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r])$$

is a valid transition in  $F$ . Since the transition  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  is not by transition case 3.2, it must be by transition case 3.1 of  $T$ . Thus,  $T$  finds a summary  $(q^{i+1}, \mathcal{M}^i[r], \mathcal{T}_{[r \rightarrow \varepsilon]}^i, \mathcal{M}_{[r \rightarrow 0]}^i, \mathcal{S}) \in$

$\mathcal{S}^i[r]$  such that for each stack  $j \neq r$ ,  $\mathcal{S}[j] \subseteq \mathcal{S}^i[j]$ . By construction of  $F$ , there is the transition

$$\begin{aligned} & (\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r S_{\uparrow}^{t+1}) \\ & \mapsto (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}_{[r \mapsto \mathcal{T}^t[r]]}^i, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}_{[r \mapsto \mathcal{S}^t[r]]}, \mathcal{M}^t[r]) \\ & \stackrel{!}{=} (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r]) \end{aligned}$$

To find the last equation, we need to identify  $\mathcal{T}_{[r \mapsto \mathcal{T}^t[r]]}^i = \mathcal{T}^{i+1}$  and  $\mathcal{S}_{[r \mapsto \mathcal{S}^t[r]]} = \mathcal{S}^{i+1}$ . Remember that  $\tau_i = (q^i, s, r, q^{i+1})$  is a popping transition with

$$\eta_i = (q^i, \mathcal{R}^i) \xrightarrow{\tau_i} (q^{i+1}, \mathcal{R}^{i+1}) = \eta_{i+1},$$

that used transition case 3.1 of the strategy automaton conditions for a transition with the prediction  $\mathcal{S} \in {}^r S_{\uparrow}^i = \mathcal{S}^i[r]$ . Since  $(t, i)$  is a push-pop-pair, for all positions  $t+1 \leq p \leq i$ ,

$$|\mathcal{R}^t[r]| = |\mathcal{R}^{t+1}[r]| - 1 = |\mathcal{R}^i[r]| - 1 = |\mathcal{R}^{i+1}[r]| < |\mathcal{R}^p[r]|.$$

As  $T$  does not change the symbol of its tuples and by repetitive use of Lemma 13,

$$\mathcal{T}^{i+1}[r] = {}^r \gamma_{\uparrow}^{i+1} = {}^r \gamma_{\uparrow}^t = \mathcal{T}^t[r] \quad \text{and} \quad \mathcal{S}^t[r] = {}^r S_{\uparrow}^t = {}^r S_{\uparrow-1}^{t+1} = {}^r S_{\uparrow-1}^i = {}^r S_{\uparrow}^{i+1} = \mathcal{S}^{i+1}[r].$$

Since  $T$  used  $\mathcal{S}$  for its case 3.1 transition, by its construction: For all stacks  $j \neq r$ ,  $\mathcal{S}[j]^{i+1} = {}^j S_{\uparrow}^{i+1} = \mathcal{S}[j]$ .

The last transition is

$$\begin{aligned} & (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r]) \mapsto (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}', \mathcal{S}^{i+1}) \\ & \stackrel{!}{=} (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

It remains to show  $\mathcal{M}' = \mathcal{M}^{i+1}$ . For any stack  $j > r$ , by construction of  $F$  and  $T$ ,

$$\mathcal{M}^{i+1}[j] = {}^j m_{\uparrow}^{i+1} = \max\{{}^j m_{\uparrow}^i, \Omega(q^{i+1})\} = \max\{\mathcal{M}^i[j], \Omega(q^{i+1})\} = \mathcal{M}'[j].$$

For stack  $r$ , again since  $(t, i)$  is a push-pop-pair, for all positions  $t+1 \leq p \leq i$  holds:

$$|\mathcal{R}^t[r]| = |\mathcal{R}^{t+1}[r]| - 1 = |\mathcal{R}^i[r]| - 1 = |\mathcal{R}^{i+1}[r]| < |\mathcal{R}^p[r]|.$$

Repetitive use of Lemma 13 leads to  ${}^r m_{\uparrow}^t = {}^r m_{\uparrow-1}^{t+1} = {}^r m_{\uparrow-1}^i$ . Finally,

$$\begin{aligned} \mathcal{M}^{i+1}[r] &= \max\{{}^r m_{\uparrow}^i, {}^r m_{\uparrow-1}^i, \Omega(q^{i+1})\} \\ &= \max\{{}^r m_{\uparrow}^i, {}^r m_{\uparrow}^t, \Omega(q^{i+1})\} = \max\{\mathcal{M}^i[r], \mathcal{M}^t[r], \Omega(q^{i+1})\} = \mathcal{M}'[r]. \end{aligned}$$

The positions

$$(\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r S_{\uparrow}^{t+1}), \quad (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r])$$

are both not owned by Eve. The transitions are compliant with  $\sigma$ .  $\blacktriangleleft$

Let  $\sigma$  be a winning strategy for Eve in  $F$  from  $(\text{Check}, q_{\text{init}}, \varepsilon^n, 0^n, \emptyset^n)$ . We use  $T$  to derive a strategy  $\nu$  for Eve in  $G$  as described above (Lemma 14). Let there be a play  $\pi$  compliant with  $\nu$  together with its strategy automaton run  $\eta$ . Towards contradiction, assume  $\pi$  is losing for Eve. We construct a play  $\rho = \rho_0 \dots$  in  $F$  compliant with  $\sigma$  from  $\rho_0 = (\text{Check}, q_{\text{init}}, \varepsilon^n, 0^n, \emptyset^n)$  that is losing for Eve.

## 52:24 On the Complexity of Multi-Pushdown Games

For each position  $i \in \mathbb{N}$ , let

$$\eta_i = (q^i, \overline{\mathcal{R}}^i), \quad \text{Tr}(\eta_i) = (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i).$$

where for each stack  $j$ ,  $\mathcal{R}[j]^i = ({}^j\gamma_{\uparrow}^i, {}^j m_{\uparrow}^i, {}^j \mathcal{S}_{\uparrow}^i)({}^j\gamma_{\uparrow-1}^i, {}^j m_{\uparrow-1}^i, {}^j \mathcal{S}_{\uparrow-1}^i) \dots ({}^j\gamma_1^i, {}^j m_1^i, {}^j \mathcal{S}_1^i)$ .

In the following, assume that for all  $i \in \mathbb{N}$  there is no transition  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  following transition case 3.2 of  $T$  for having a transition. We handle that case later.

► **Lemma 16.** *Let  $p \in \mathbb{N}$  be a stair of  $\eta$ . There is  $\psi : \mathbb{N} \rightarrow \mathbb{N}$ , such that for each  $i \in \mathbb{N}$  with  $p \leq i$ , there is a play  $\rho^i = \rho_1^i \mapsto \dots \mapsto \rho_{\psi(i)}^i$  of length  $\psi(i)$  compliant with  $\sigma$  in  $F$  from  $\rho_1^i = \text{Tr}(\eta_p)$  to  $\rho_{\psi(i)}^i = \text{Tr}(\eta_i)$  such that*

$$\max_{u \in [p..i]} \{\Omega(q^u)\} = \max_{u \in [1..\psi(i)]} \{\Omega(\rho_u)\}.$$

**Proof.** We show this by induction.

► **Base Case** ( $i = p$ ). Set  $\psi(p) = 1$ . Immediately,  $\text{Tr}(\eta_p) = \text{Tr}(\eta_i)$  and  $\Omega(q^p) = \Omega(\text{Tr}(\eta_p))$ .

► **Inductive Case** ( $i \mapsto i + 1$ ). Assume, that for each  $t \in [p..i]$ , there is a play  $\rho^t$  compliant with  $\sigma$  from  $\text{Tr}(\eta_p) = \rho_1^t$  to  $\text{Tr}(\eta_t) = \rho_{\psi(t)}^t$ .

We create a play in  $F$  from  $\text{Tr}(\eta_p)$  to  $\text{Tr}(\eta_{i+1})$  compliant with  $\sigma$ .

**Case 1** ( $\tau_i \in \delta_{int}$ ): Set  $\psi(i + 1) = \psi(i) + 1$ . The desired play is a continuation of  $\rho^i$ . By Lemma 15, the following transition is compliant with  $\sigma$ .

$$\text{Tr}(\eta_i) = (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \mapsto (\text{Check}, q^{i+1}, \mathcal{T}, \mathcal{M}^{i+1}, \mathcal{S}) = \text{Tr}(\eta_{i+1})$$

By induction,

$$\begin{aligned} \max_{u \in [p..i+1]} \{\Omega(q^u)\} &= \max\left\{ \max_{u \in [p..i]} \{\Omega(q^u)\}, \Omega(q^{i+1}) \right\} \\ &= \max\left\{ \max_{u \in [1..\psi(i)]} \{\Omega(\rho_u^i)\}, \Omega(\rho_{\psi(i+1)}^i) \right\} = \max_{u \in [1..\psi(i+1)]} \{\Omega(\rho_u^{i+1})\} \end{aligned}$$

**Case 2** ( $\tau_i = (q^i, r, s, q^{i+1}) \in \delta_{push}$ ): The desired play is a continuation of  $\rho^i$ , which ends in  $\text{Tr}(\eta_i)$ . By Lemma 15, the following transitions are compliant with  $\sigma$ .

$$\begin{aligned} \text{Tr}(\eta_i) &= (\text{Check}, q^i, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i) \\ &\mapsto (\text{Push}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^i, \mathcal{M}^i, \mathcal{S}^i, q^{i+1}, s, {}^r \mathcal{S}_{\uparrow}^{i+1}) \\ &\mapsto (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = \text{Tr}(\eta_{i+1}) \end{aligned}$$

By induction,

$$\begin{aligned} \max_{u \in [p..i+1]} \{\Omega(q^u)\} &= \max\left\{ \max_{t \in [p..i]} \{\Omega(q^t)\}, \Omega(q^{i+1}) \right\} \\ &= \max\left\{ \max_{u \in [1..\psi(i)]} \{\Omega(\rho_u^p)\}, 0, 0, \Omega(q^{i+1}) \right\} \\ &= \max\left\{ \max_{u \in [1..\psi(i)]} \{\Omega(\rho_u^p)\}, \Omega(\text{Push}_r, -), \Omega(\text{Claim}_r, -), \right. \\ &\quad \left. \Omega(\text{Check}, q^{i+1}, -) \right\} \\ &= \max_{u \in [1..\psi(i+1)]} \{\Omega(\rho_u^p)\} \end{aligned}$$

**Case 3** ( $\tau_i = (q^i, s, r, q^{i+1}) \in \delta_{pop}$ ): Since  $p$  is a stair, position  $i$  is in a push-pop-pair  $(t, i)$  such that  $p \leq t < i$ . Set  $\psi(i+1) = \psi(t) + 4$ . The desired play is a continuation of  $\rho^t$ . Because  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  is not by transition case 3.2 of  $T$ 's transition conditions, by Lemma 15, the following transitions are compliant with  $\sigma$ .

$$\begin{aligned} Tr(\eta_t) &= (\text{Check}, q^t, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t) \\ &\mapsto (\text{Push}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}^t, \mathcal{M}^t, \mathcal{S}^t, q^{t+1}, s, {}^r\mathcal{S}_\uparrow^{t+1}) \\ &\mapsto (\text{Jump}_r, q^{i+1}, \mathcal{M}^i[r], \mathcal{T}^{i+1}, \mathcal{M}_{[r \rightarrow 0]}^i, \mathcal{S}^{i+1}, \mathcal{M}^t[r]) \\ &\mapsto (\text{Check}, q^{i+1}, \mathcal{T}^{i+1}, \mathcal{M}^{i+1}, \mathcal{S}^{i+1}) = Tr(\eta_{i+1}) \end{aligned}$$

Since  $(t, i)$  is push-pop-pair,  $\text{lup}_r^\eta(i) = t$ . By Lemma 14,

$$\mathcal{M}^i[r] = \max_{u \in [\text{lup}_r^\eta(i) .. i]} \{\Omega(q^u)\} = \max_{u \in [t .. i]} \{\Omega(q^u)\}.$$

By induction,

$$\begin{aligned} \max_{u \in [p .. i+1]} \{\Omega(q^u)\} &= \max \left\{ \max_{u \in [p .. t]} \{\Omega(q^u)\}, \max_{u \in [t .. i]} \{\Omega(q^u)\}, \Omega(q^{i+1}) \right\} \\ &= \max \left\{ \max_{u \in [1 .. \psi(t)]} \{\Omega(\rho_u^t)\}, \mathcal{M}[r]^i, \Omega(\rho_{\psi(i+1)}^t) \right\} \\ &= \max_{u \in [1 .. \psi(i+1)]} \{\Omega(\rho_u^{i+1})\} \quad \blacktriangleleft \end{aligned}$$

Since  $(\mathbb{N}^n, \leq_n)$  is a well-quasi ordering,  $\eta$  contains an infinite set of stairs  $\mathcal{ST}_\eta = \{p_1, p_2, \dots\} \subseteq \mathbb{N}$  with  $p_1 < p_2 < \dots$ . Towards contradiction, we can now construct a play  $\rho = \rho_0 \mapsto \rho_1 \mapsto \dots$  in  $F$  that is winning for Ana and is compliant with  $\sigma$ . We need a function  $\phi : \mathcal{ST} \rightarrow \mathbb{N}$ , such that for any  $p \in \mathcal{ST}$ ,  $Tr(\eta_p) = \rho_{\phi(p)}$ . Furthermore, we want for each  $p_i, p_{i+1} \in \mathcal{ST}$  that

$$\max_{u \in [p_i .. p_{i+1}]} \{\Omega(\pi_u)\} = \max_{u \in [\phi(p_i) .. \phi(p_{i+1})]} \{\Omega(\rho_u)\},$$

which leads to

$$\begin{aligned} \max_{u \in \mathbb{N}} \inf \{\Omega(\pi_u)\} &= \max_{i \in \mathbb{N}} \inf \left\{ \Omega(\pi_{p_i}), \max_{p_i < u < p_{i+1}} \Omega(\pi_u) \right\} = \\ &= \max_{i \in \mathbb{N}} \inf \left\{ \Omega(\rho_{\phi(p_i)}), \max_{\phi(p_i) < u < \phi(p_{i+1})} \Omega(\rho_u) \right\} = \max_{u \in \mathbb{N}} \inf \{\Omega(\rho_u)\}. \end{aligned}$$

And thus  $\rho$  is a play compliant with  $\sigma$ , that is won by Ana, contradicting  $\sigma$  being a winning strategy for Eve.

► **Base Case** ( $p_1$ ). Initial position of the play is  $\rho_0 = (\text{Check}, q_{init}, \varepsilon^n, 0^n, \emptyset^n) = Tr(\eta_0)$ , which is a stair. Thus,  $p_1 = 0$ .

► **Inductive Case** ( $p_i \rightarrow p_{i+1}$ ). Assume, we constructed  $\rho$  and the function  $\phi$ , such that

$$\rho_0 = Tr(\eta_0) \mapsto \dots \mapsto \rho_{\phi(p_1)} = Tr(\eta_1) \mapsto \dots \mapsto \rho_{\phi(p_2)} = Tr(\eta_2) \mapsto \dots \mapsto \rho_{\phi(p_i)} = Tr(\eta_{p_i})$$

and for all  $i \in [1 .. i-1]$ ,

$$\max_{u \in [p_i .. p_{i+1}]} \{\Omega(\pi_u)\} = \max_{u \in [\phi(p_i) .. \phi(p_{i+1})]} \{\Omega(\rho_u)\}.$$

Since  $\eta_{p_i}$  is a stair and  $Tr(\eta_{p_i}) = \rho_{\phi(p_i)}$ , by Lemma 16, we can find a position for  $\phi(p_{i+1})$  and continue  $\rho$  by some transitions  $Tr(\eta_{p_i}) = \rho_{\phi(p_i)} \mapsto \dots \mapsto \rho_{\phi(p_{i+1})} = Tr(\eta_{p_{i+1}})$ , such that

$$\max_{u \in [p_i..p_{i+1}]} \{\Omega(\pi_u)\} = \max_{u \in [\phi(p_i)..\phi(p_{i+1})]} \{\Omega(\rho_u)\}.$$

Now we handle the case where one of the transitions in  $\eta$  is due to transition case 3.2 of the strategy automaton. Let there is a minimal position  $i \in \mathbb{N}$ , such that  $\eta_i \xrightarrow{\tau_i} \eta_{i+1}$  is due to transition case 3.2 of  $T$ 's transition conditions. Be aware, that the induction in Lemma 16 still works up to position  $i$ . Thus, there is a play  $\rho$  compliant with  $\sigma$  from (Check,  $q_{init}, \varepsilon^n, 0^n, \emptyset^n$ ), which is a stair, to  $\psi(\eta_i)$ . Since  $T$  had a transition for  $\tau_i$ , either  $own(\psi(\eta_i)) = \text{Ana}$  or  $\sigma$  chose the transition introduced to  $F$  caused by  $\tau_i$ . In either case, the following transition is compliant with  $\sigma$ :

$$\psi(\eta_i) = (\text{Check}, q^i, \mathcal{T}^i, \mathcal{S}^i, \mathcal{M}^i) \mapsto \text{AnaWin}.$$

Because  $\tau_i$  used transition case 3.2, we know that there is no  $(q^{i+1}, \mathcal{M}^i[r], \mathcal{T}_{[r \mapsto \varepsilon]}^i, \mathcal{M}_{[r \mapsto 0]}^i, \mathcal{S}) \in \mathcal{S}[r]$  such that for each stack  $j > r$ ,  $\mathcal{S}[j] \subseteq {}^j\mathcal{S}_\uparrow$ .

Thus the above transition is indeed a continuation of  $\rho$ , compliant with  $\sigma$ , that is won by Ana, contradicting  $\sigma$  being a winning strategy for Eve.

## B.2 Transforming a winning strategy from $G$ to $F$

We handle a lot of play prefixes in this section. Let us introduce the notation  $\pi_{..i} = \pi_0 \pi_1 \dots \pi_i$  for play prefixes of  $\pi$ .

Let  $\nu$  be a winning strategy for Eve in  $G$ . We construct a strategy  $\sigma$  for Eve in  $F$ . For this, we need to maintain a play prefix of  $G$ . During a play  $\rho$  in  $F$ , we build up and continue this prefix and use it to determine the moves to be taken by  $\sigma$  in  $\rho$ .

► **Definition 17.** *Given a strategy  $\nu$  for Eve in  $G$ , a play prefix  $\pi_{..l} = \pi_0 \xrightarrow{\tau_0} \dots \xrightarrow{\tau_{l-1}} \pi_l$  compliant with  $\nu$  and an unmatched pushing position  $p \in [0..l-1]$  with  $\tau_p = (q, r, s, q') \in \delta_{push}$ , we define the summary set  $S^{\pi_{..l}, \nu, p} \subseteq \mathbb{OS}_r$  recursively:*

*For every play  $\pi_{..l'}$  that is a continuation of  $\pi_{..l}$ , i.e.  $l < l'$ , and compliant with  $\nu$ , if  $p$  is matched in  $\pi_{..l'}$ , i.e.  $(p, t)$  is a push-pop-pair in  $\pi_{..l}$ , we add a summary as follows to  $S^{\pi_{..l}, \nu, p}$ :*

*Let  $q''$  be the state at position  $t+1$ , and  $\mathcal{T}$  be the top of stack symbols at  $\pi_{t+1}$ . For each stack  $j \in [1..n]$ , let  $t_j = \text{lup}_j^{\pi_{..t}}(t)$ . Let  $\mathcal{M}$  be such that*

$$\mathcal{M}[j] = \begin{cases} \max_{u \in [t_j+1..t]} \{\Omega(\pi_u)\} & t_j \neq \perp \\ \max_{u \in [0..t]} \{\Omega(\pi_u)\} & t_j = \perp \end{cases}$$

*We add  $(q'', \mathcal{M}[r], \mathcal{T}_{[r \mapsto \varepsilon]}, \mathcal{M}_{[r \mapsto 0]}, \mathcal{S})$  to  $S^{\pi_{..l}, \nu, p}$ , where  $\mathcal{S}[j]$  for each stack  $j > r$ ,*

$$\mathcal{S}[j] = S^{\pi_{..t+1}, \nu, t_j}.$$

Be aware, that this construction is finite and the result is an actual prediction: The sets  $S^{\pi_{..t+1}, \nu, t_j}$  contain sets of summaries for only stacks greater, thus the recursion terminates for stack  $n$ . Furthermore, this construction is finite as  $\mathbb{OS}_j$  is finite.

For a play prefix  $\pi_{..l}$  and its continuation  $\pi_{..l'}$ , i.e.  $l < l'$ , it is immediate that  $S^{\pi_{..l'}, \nu, p} \subseteq S^{\pi_{..l}, \nu, p}$ . This is because the set of play continuations for  $\pi_{..l'}$  is a subset of the play continuations for  $\pi_{..l}$ .

For a play  $\rho$  in  $F$  compliant with  $\sigma$ , we maintain the play prefix of  $G$ . In order to keep the construction short, we define the strategy and an invariant (Lemma 18) between the

two plays at the same time. For this, define the set  $\text{Checks}^\rho \subseteq \mathbb{N}$  of all indices where  $\rho$  is in a Check-state. We can order these positions by their occurrence in  $\rho$  so we get  $\text{Checks}^\rho = \{p_1, p_2, \dots\}$  with  $p_1 < p_2 < \dots$ . We define a function  $\psi : \text{Checks}^\rho \rightarrow \mathbb{N}$  that maps play indices of Check-states in  $\rho$  to positions in  $\pi$ .

► **Lemma 18.** *Let  $\rho$  be a play compliant with  $\sigma$  and  $\pi$  the corresponding play created.*

- $\pi$  is compliant with  $\nu$
- For any position  $p \in \text{Checks}^\rho$ , if  $\rho_p = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$ , then  $\pi_{\psi(p)}$  is in state  $q$  and the top of stack symbols are  $\mathcal{T}$ . Let  $t_j = \text{lup}_j^{\pi.. \psi(p)}(\psi(p))$ . For every stack  $j$  with  $t_j \neq \perp$ ,  $S^{\pi.. \psi(p), \nu, t_j} \subseteq \mathcal{S}[j]$ .
- for  $p_i, p_{i+1} \in \text{Checks}^\rho$ ,

$$\max_{u \in [p_i..p_{i+1}]} \{\Omega(\rho_u)\} = \max_{u \in [\psi(p_i).. \psi(p_{i+1})]} \{\Omega(\pi_u)\}$$

**Proof and Construction.** by induction.

► **Base Case** ( $i = 1$ ). This is only the initial position.  $\pi_0 = (q_{init}, \varepsilon^n)$ ,  $\rho_0 = (\text{Check}, q_{init}, \varepsilon^n, \Omega(q_{init})^n, \emptyset^n)$ .  $\psi(p_1) = \psi(0) = 0$ .

► **Inductive Case** ( $i \rightarrow i + 1$ ). We first show how  $\sigma$  continues  $\rho$  and  $\pi.. \psi(p_i)$  before focussing on the invariant stated in Lemma 18.

If  $\text{own}(\rho_{p_i}) = \text{Eve}$ , we need to construct  $\sigma$  for  $\rho_{p_i} = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S})$ . In that case, let  $\pi_{\psi(p_i)} \xrightarrow{\tau} \nu(\pi_{\psi(p_i)})$  be the transition used by  $\nu$  in  $G$ . If  $\text{own}(\rho_{p_i}) = \text{Ana}$ , Ana takes some transition in  $F$  that was introduced by some transition  $\tau$  enabled in  $\pi_{\psi(p_i)}$ .

Let  $\pi_{\psi(p_i)} = (q, \mathcal{P})$ , where by induction, the top of stack symbols form  $\mathcal{T}$ .

**Case 1** ( $\tau = (q, r, q') \in \delta_{int}$ ):  $\rho$  continues with the transition introduced in  $F$ :

$$\rho_{p_i} = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \xrightarrow{\tau} (\text{Check}, q', \mathcal{T}, \mathcal{M}', \mathcal{S}) = \rho_{p_{i+1}}$$

Further, we set  $\psi(p_{i+1}) = \psi(p_i) + 1$  and continue  $\pi$  by

$$\pi_{\psi(p_i)} = (q, \mathcal{P}) \xrightarrow{\tau} (q', \mathcal{P}) = \pi_{\psi(p_{i+1})}.$$

to arrive at  $\pi.. \psi(p_{i+1})$ .

To the invariant: This transition is compliant with  $\nu$ . The state conditions are fulfilled by construction, as well as the top of stack condition. The prediction sets did not change, thus  $S^{\pi.. \psi(p_{i+1}), \nu, t_j} \subseteq S^{\pi.. \psi(p_i), \nu, t_j} \subseteq \mathcal{S}[j]$ . And for the parity condition,

$$\begin{aligned} \max_{u \in [p_i..p_{i+1}]} \{\Omega(\rho_u)\} &= \max\{\Omega(\rho_{p_i}), \Omega(\rho_{p_{i+1}})\} = \\ &= \max\{\Omega(\pi_{\psi(p_i)}), \Omega(\pi_{\psi(p_{i+1})})\} = \max_{u \in [\psi(p_i).. \psi(p_{i+1})]} \{\Omega(\pi_u)\}. \end{aligned}$$

**Case 2** ( $\tau = (q, r, s, q') \in \delta_{push}$ ):  $\rho$  continues with the transition introduced in  $F$ :

$$\rho_{p_i} = (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \mapsto (\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s)$$

First, we continue  $\pi.. \psi(p_i)$  by  $\pi.. \psi(p_i) \xrightarrow{\tau} \pi_{\psi(p_i)+1}$  which is compliant with  $\nu$ .

Then, Eve has to make a claim in  $F$ . To define their strategy, we use the game prediction from above.

$$\sigma(\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s) = (\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s, S^{\pi.. \psi(p_i)+1, \nu, \psi(p_i)})$$

*Case 2.1* (Ana continues to  $(\text{Check}, q', r, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}', \mathcal{S}_{[r \rightarrow S^{\pi.. \psi(p_i)+1, \nu, \psi(p_i)}]})$ ): The transitions in  $F$  up to  $p_{i+1}$  are

$$\begin{aligned} \rho_{p_i} &= (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \\ &\mapsto (\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s, S^{\pi.. \psi(p_i)+1, \nu, \psi(p_i)}) \\ &\mapsto (\text{Check}, q', r, \mathcal{T}_{[r \rightarrow s]}, \mathcal{M}', \mathcal{S}_{[r \rightarrow S^{\pi.. \psi(p_i)+1, \nu, \psi(p_i)}]}) = \rho_{p_{i+1}} \end{aligned}$$

Thus,  $p_{i+1} = p_i + 3$ . Set  $\psi(p_{i+1}) = \psi(p_i) + 1$ . Then,  $\pi.. \psi(p_{i+1}) = \pi.. \psi(p_i) + 1$ .

To the invariant, state conditions are fulfilled by construction, as well as the top of stack condition. The prediction sets for all stacks  $j \neq r$  did not change and  $t_j = \text{lup}_j^{\pi.. \psi(p_i)}(\psi(p_i)) = \text{lup}_j^{\pi.. \psi(p_{i+1})}(\psi(p_{i+1}))$ , thus  $S^{\pi.. \psi(p_{i+1}), \nu, t_j} \subseteq S^{\pi.. \psi(p_i), \nu, t_j} \subseteq S[j] = \mathcal{S}_{[r \rightarrow S^{\pi.. \psi(p_i), \nu, \psi(p_i)}]}[j]$ .

For stack  $r$ , we have  $\text{lup}_r^{\pi.. \psi(p_{i+1})}(\psi(p_{i+1})) = \psi(p_i)$ . Adequately,  $S^{\pi.. \psi(p_{i+1}), \nu, \psi(p_i)} = \mathcal{S}_{[r \rightarrow S^{\pi.. \psi(p_i), \nu, \psi(p_i)}]}[r]$ .

For the parity condition, we have

$$\begin{aligned} \max_{u \in [p_i.. p_{i+1}]} \{\Omega(\rho_u)\} &= \max\{\rho_{p_i}, (\text{Push}_r, -), (\text{Claim}_r, -), \rho_{p_{i+1}}\} = \\ &= \max\{\rho_{p_i}, \rho_{p_{i+1}}\} = \max\{\pi_{\psi(p_i)}, \pi_{\psi(p_{i+1})}\} = \max_{u \in [\psi(p_i).. \psi(p_{i+1})]} \{\Omega(\pi_u)\}. \end{aligned}$$

*Case 2.2* (Ana continues to  $(\text{Jump}_r, q'', m, \mathcal{T}'_{[r \rightarrow \mathcal{T}[r]]}, \mathcal{M}', \mathcal{S}'_{[r \rightarrow S[r]]}, \mathcal{M}[r])$ ): The transitions in  $F$  up to  $p_{i+1}$  are

$$\begin{aligned} \rho_{p_i} &= (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \\ &\mapsto (\text{Push}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s) \\ &\mapsto (\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s, S^{\pi.. \psi(p_i), \nu, \psi(p_i)}) \\ &\mapsto (\text{Jump}_r, q'', m, \mathcal{T}'_{[r \rightarrow \mathcal{T}[r]]}, \mathcal{M}', \mathcal{S}'_{[r \rightarrow S[r]]}, \mathcal{M}[r]) \\ &\mapsto (\text{Check}, q'', \mathcal{T}'_{[r \rightarrow \mathcal{T}[r]]}, \mathcal{M}'', \mathcal{S}'_{[r \rightarrow S[r]]}) = \rho_{p_{i+1}} \end{aligned}$$

We set  $p_{i+1} = p_i + 4$ . Further, it must be that  $(q'', m, \mathcal{T}', \mathcal{M}', \mathcal{S}') \in S^{\pi.. \psi(p_i), \nu, \psi(p_i)}$  in order for

$$(\text{Claim}_r, \mathcal{T}, \mathcal{M}, \mathcal{S}, q', s, S^{\pi.. \psi(p_i), \nu, \psi(p_i)}) \mapsto (\text{Jump}_r, q'', m, \mathcal{T}'_{[r \rightarrow \mathcal{T}[r]]}, \mathcal{M}', \mathcal{S}'_{[r \rightarrow S[r]]}, \mathcal{M}[r])$$

to exist. By construction of  $S^{\pi.. \psi(p_i), \nu, \psi(p_i)}$ , there is a play continuation  $\pi.. l$  of  $\pi.. \psi(p_i)$  compliant with  $\nu$ , such that  $\psi(p_i)$  is in a push-pop-pair  $(\psi(p_i), t)$  with  $\psi(p_i) < t < l$ .

Finally, we continue  $\pi.. \psi(p_i)$  to  $\pi.. t+1$  and set  $\psi(p_{i+1}) = t + 1$ .

To the invariant: By construction of  $S^{\pi.. \psi(p_i), \nu, \psi(p_i)}$ , this is compliant with  $\nu$ .

By construction of  $S^{\pi.. \psi(p_i), \nu, \psi(p_i)}$ ,  $\pi_{t+1}$  is in state  $q''$  with the top of stack symbols being  $\mathcal{T}'_{[r \rightarrow \gamma[r]]}$ .

Furthermore, for each stack  $j > r$ , since  $(q'', m, \mathcal{T}', \mathcal{M}', \mathcal{S}') \in S^{\pi.. \psi(p_i), \nu, \psi(p_i)}$ , with  $t_j = \text{lup}_j^{\pi.. t}(t)$ ,

$$\mathcal{S}'_{[r \rightarrow S[r]]}[j] = \mathcal{S}'[j] = \begin{cases} S^{\pi.. t, \nu, t_j} & t_j \neq \perp \\ \emptyset & t_j = \perp \end{cases}.$$

For stack  $r$ , we know that since  $(\psi(p_i), t)$  is a push-pop-pair, that

$$t_r = \text{lup}_r^{\pi.. \psi(p_i)}(\psi(p_i)) = \text{lup}_r^{\pi.. t+1}(t + 1) = \text{lup}_r^{\pi.. \psi(p_{i+1})}(\psi(p_{i+1})).$$

Due to  $\pi_{.. \psi(p_{i+1})}$  being a continuation of  $\pi_{.. \psi(p_i)}$ , we arrive at

$$S^{\pi_{.. \psi(p_{i+1})}, \nu, t_r} \subseteq S^{\pi_{.. \psi(p_i)}, \nu, t_r} \subseteq \mathcal{S}[r] = \mathcal{S}'_{[r \mapsto \mathcal{S}[r]]}[r].$$

For the parity condition, be aware, that by construction of  $S^{\pi_{.. \psi(p_i)}, \nu, \psi(p_i)}$ ,

$$m = \max_{u \in [\psi(p_i)+1..t]} \{\Omega(\pi_u)\}.$$

Together, we arrive at:

$$\begin{aligned} \max_{u \in [p_i..p_{i+1}]} \{\Omega(\rho_u)\} &= \max\{\rho_{p_i}, \rho_{p_{i+1}}, (\text{Push}_r, -), (\text{Claim}_r, -), \\ &\quad (\text{Jump}_r, q'', m, \mathcal{T}'_{[r \mapsto \mathcal{T}[r]]}, \mathcal{M}', \mathcal{S}'_{[r \mapsto \mathcal{S}[r]]}, \mathcal{M}[r])\} \\ &= \max\{\rho_{p_i}, \rho_{p_{i+1}}, m\} = \max_{u \in [\psi(p_i).. \psi(p_{i+1})]} \{\Omega(\pi_u)\}. \end{aligned}$$

**Case 3** ( $\tau = (q, s, r, q') \in \delta_{pop}$ ):  $\rho$  continues with the transition introduced for  $\tau$ . This is either

$$\begin{aligned} \rho_{p_i} &= (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \mapsto \text{EveWin} \quad \text{or} \\ \rho_{p_i} &= (\text{Check}, q, \mathcal{T}, \mathcal{M}, \mathcal{S}) \mapsto \text{AnaWin}. \end{aligned}$$

We show, that the second case is impossible. Since  $\tau$  is enabled in  $\pi_{\psi(p_i)}$ , we can continue  $\pi_{.. \psi(p_i)}$  by  $\pi_{\psi(p_i)} \xrightarrow{\tau} \pi_{\psi(p_i)+1}$ . Due to the enabledness of a pop-transition, there is  $t_r = \text{lup}_r^{\pi_{.. \psi(p_i)}}(\psi(p_i))$  and by definition of  $S^{\pi_{.. \psi(p_i)}, \nu, t_r}$ , there is a summary  $(q', \mathcal{M}[r], \mathcal{T}_{[r \mapsto \varepsilon]}, \mathcal{M}_{[r \mapsto 0]}, \mathcal{S}') \in S^{\pi_{.. \psi(p_i)}, \nu, t_r} \subseteq \mathcal{S}[r]$ , such that for all stacks  $j > r$ :

$$\mathcal{S}'[j] = S^{\pi_{.. \psi(p_i)+1}, \nu, t_j} \subseteq \mathcal{S}[j].$$

Thus, by construction of  $F$ , the second transition does not exist.

Now we can show, that  $\rho$  is winning for Eve:

**Case 1** ( $\rho$  contains EveWin): This play is winning for Eve.

**Case 2** ( $\rho$  contains AnaWin): We have just shown, that this can not happen.

**Case 3** ( $\rho$  contains infinitely many Check states): In this case, Checks $^\rho$  is infinite and

$$\begin{aligned} \max_{u \in \mathbb{N}} \{\Omega(\rho_u)\} &= \max_{i \in \mathbb{N}} \left\{ \max_{u \in [p_i..p_{i+1}]} \{\Omega(\rho_u)\} \right\} \\ &= \max_{i \in \mathbb{N}} \left\{ \max_{u \in [\psi(p_i).. \psi(p_{i+1})]} \{\Omega(\pi_u)\} \right\} = \max_{u \in \mathbb{N}} \{\Omega(\pi_u)\}, \end{aligned}$$

which is winning for Eve, since  $\pi$  is compliant with  $\nu$ , which is a winning strategy.  $\blacktriangleleft$

## C Stack Elimination for Context-Bounded MPDG

For  $k$ -context bounded MPDG can only visit up to  $k$  stacks in a play, we can eliminate stacks to obtain a  $k$ -context  $k$ -stack MPDG.

► **Lemma 19.** *For every  $k$ -context-bounded  $n$ -stack MPDG  $G = (P, \text{own}, \Omega)$  with MPDS  $P = (Q, \Gamma, \delta, n)$ , there is  $k$ -context-bounded  $k$ -stack MPDG  $G' = (P', \text{own}', \Omega')$  with  $P' = (Q', \Gamma, \delta', k)$  such that Eve wins  $G$  if and only if she wins  $G'$ . The set of priorities coincide and  $G'$  is constructible in time  $\mathcal{O}(|G| \cdot n^{k+1})$ .*

## 52:30 On the Complexity of Multi-Pushdown Games

First, present the construction of  $P'$ : The new state space is  $Q' = [1..n]^k \times [0..k] \times [1..n] \times [1..k] \times Q$ . A configuration of  $P'$  is thus  $((f, k, d, e, q), \mathcal{R})$ , where the task of the different parameters is as follows.  $f$  is an injective mapping from the available stacks  $[1..k]$  to the used stacks of  $P$ . The inverse function is  $f^{-1}$ .  $k$  tracks the current context.  $d$  tracks the stack of the current context.  $e$  tracks the number of different stacks used so far.  $\mathcal{R} : [1..k] \rightarrow \Gamma^*$  are the stack contents.

For each transition  $\tau \in \delta_{d'}$  with  $(q, \mathcal{P}) \xrightarrow{\tau} (q', \mathcal{P}')$ ,  $P'$  has transitions

$$((f, k, d, e, q), \mathcal{R}) \xrightarrow{\tau} ((f', k', d', e', q), \mathcal{R}_{[(f')^{-1}(d') \mapsto \mathcal{P}'[d']]}),$$

where either

- $d' = d$  and  $f' = f$ ,  $k' = k$ ,  $e' = e$  or
- $d \neq d'$  and  $k + 1 \leq k'$  and  $k' = k + 1$  and  $f^{-1}(d') \neq \perp$  and  $e' = e$  and  $f' = f$  or
- $d \neq d'$  and  $k + 1 \leq k'$  and  $k' = k + 1$  and  $f^{-1}(d') = \perp$  and  $e' = e + 1$  and  $f' = f_{[e' \mapsto d']}$ .

Let  $\pi = \pi_0 \rightarrow \dots \rightarrow \pi_l$  be a play prefix of  $G$ . We define the function  $g_\pi$ , which takes a position  $p$  of the run  $\pi$  and transforms it to stack contents  $\mathcal{R}^p$  for  $P'$ . It takes the stack contents of  $\pi_p = (q^p, \mathcal{P}^p)$  and reduces them to the stacks to which a transition belonged in  $\pi_0 \rightarrow \dots \rightarrow \pi_p$ , then reorders them, so that they are in the order in which the stacks were visited with their first respective context. Further, let  $f^p$  be the corresponding stack assigning function,  $e^p$  the number of stacks visited so far,  $k^p$  the context, and  $d^p$  the stack of that context at position  $p$ . Thus, for all already visited stacks  $d$ ,  $\mathcal{P}^p[d] = \mathcal{R}^p[f(d)]$ . Define the function  $h(\pi) = ((f^0, k^0, d^0, e^0, q^0), g(0)) \rightarrow \dots \rightarrow ((f^l, k^l, d^l, e^l, q^l), g(l))$

Vice versa, let  $\pi' = \pi'_0 \rightarrow \dots \rightarrow \pi'_l$  be a play prefix of  $G'$ . We create the function  $h'(\pi') = (q^0, \mathcal{P}^0) \rightarrow \dots \rightarrow (q^l, \mathcal{P}^l)$ , where for every position  $p$  and stack  $j$ ,

$$\mathcal{P}^p[j] = \begin{cases} \varepsilon & (f^p)^{-1}(j) = \perp \\ \mathcal{R}^p[(f^p)^{-1}(j)] & \text{otherwise.} \end{cases}$$

► **Lemma 20.**  *$h$  and  $h'$  form a bijection on the play prefixes starting with empty stack contents, i.e.  $h'(h(\pi)) = \pi$  and  $h(h'(\pi')) = \pi'$  for all play prefixes  $\pi$  and  $\pi'$  starting with empty stacks.*

**Proof.** By induction on the length  $l$  of the plays.

**Base Case.** At  $\pi_0$ , no stacks were visited. Thus,  $f$  is undefined, no context has been visited and there is no active stack, and no stacks have been used. Rerversely, at  $\pi'_0$ ,  $f$  is undefined for every stack. Thus,  $h'(h(\pi)) = \pi = \pi_0$  and  $h(h'(\pi')) = \pi' = \pi'_0$ .

**Ind. Case.** Let  $\pi = \pi_0 \rightarrow \dots \rightarrow \pi_l \xrightarrow{\tau} \pi_{l+}$  with  $h'(h(\pi_0 \dots \pi_l)) = \pi_0 \dots \pi_l$  with  $h(\pi_0 \dots \pi_l) = \pi'_0 \rightarrow \dots \rightarrow \pi'_l$ . Then,  $\pi'_l = ((f^l, k^l, d^l, e^l, q^l), \mathcal{R}^l)$ , where  $\mathcal{R}^l = g_\pi(l)$ .

We have  $h(\pi) = h(\pi_0 \dots \pi_l) \rightarrow \pi'_{l+1}$ , where  $\pi'_{l+1} = ((f^{l+1}, k^{l+1}, d^{l+1}, e^{l+1}, q^{l+1}), \mathcal{R}^{l+1})$  and  $\mathcal{R}^{l+1} = g_\pi(l)$ .

*Case 1.*  $\tau \in \delta_{d^p}$ . Then, there is the transition to  $((f', k', d', e', q), \mathcal{R}_{[(f')^{-1}(d') \mapsto \mathcal{P}'[d']]}),$  where  $d' = d^p$  and  $f' = f^p$ ,  $k' = k^p$ ,  $e' = e^p$ . Since the stack did not change, these coincide with  $d^{p+1}$ ,  $f^{p+1}$ ,  $k^{p+1}$ , and  $e^{p+1}$ . The stack contents did also change for the stack representing stack  $d^p$  by  $\mathcal{R}_{[(f^p)^{-1}(d^p) \mapsto \mathcal{P}'[d^p]]}$ . Thus,  $h(\pi)$  is a play prefix in  $G'$ .

*Case 2.*  $\tau \in \delta_{d^{p+1}}$  and  $d^{p+1} \neq d^p$  and  $(f^p)^{-1}(d^{p+1}) = \perp$ . To be  $k$ -bounded,  $k^p$  must be less than  $k$ . Then, there is the transition to  $((f', k', d', e', q), \mathcal{R}_{[(f')^{-1}(d') \mapsto \mathcal{P}'[d']]}),$  where  $e' = e^p + 1$  and  $f' = f_{[e' \mapsto d']}$ . Since the stack was not seen before (by induction, it was not found in  $f^p$ ),  $f' = f^{p+1}$ , further,  $k^{p+1} = k' = k^{p+1}$ , the next context is introduced and the

active stack is  $d^{p+1} = d'$ . The stack contents did also change for the stack representing stack  $d^p$  by  $\mathcal{R}_{[(f^{p+1})^{-1}(d^{p+1}) \mapsto \mathcal{P}'[d^{p+1}]]}$ . Thus,  $h(\pi)$  is a play prefix in  $G'$ .

*Case 3.*  $\tau \in \delta_{d^{p+1}}$  and  $d^{p+1} \neq d^p$  and  $(f^p)^{-1}(d^{p+1}) \neq \perp$ . To be  $k$ -bounded,  $k^p$  must be less than  $k$ . Then, there is the transition to  $((f', k', d', e', q), \mathcal{R}_{[(f')^{-1}(d') \mapsto \mathcal{P}'[d']]}),$  where  $e' = e^p$  and  $f' = f$ . Since the stack was seen before (by induction, it was found in  $f^p$ ),  $f' = f^{p+1}$ , further,  $k^{p+1} = k' = k^{p+1}$ , the next context is introduced and the active stack is  $d^{p+1} = d'$ . The stack contents did also change for the stack representing stack  $d^p$  by  $\mathcal{R}_{[(f^{p+1})^{-1}(d^{p+1}) \mapsto \mathcal{P}'[d^{p+1}]]}$ . Thus,  $h(\pi)$  is a play prefix in  $G'$ .

Further, by induction  $h'(h(\pi)) = \pi_0 \dots \pi_l \rightarrow (q^{l+1}, \mathcal{P})$ , where for every stack  $j$ ,

$$\mathcal{P}[j] = \begin{cases} \mathcal{R}^{l+1}[(f^p)^{-1}(j)] = \mathcal{P}^{l+1}[j] & (f^p)^{-1}(j) \neq \perp \\ \varepsilon = \mathcal{P}^{l+1}[j] & (f^p)^{-1}(j) = \perp \end{cases}$$

Thus,  $h'(h(\pi)) = \pi$ .

Showing  $h(h'(\pi')) = \pi'$  is analogue.  $\blacktriangleleft$

**Proof of Lemma 19.** Together with the ownership assignment of  $own'((f, k, d, e, q), \mathcal{R}) = own(q)$  and priority assignment  $\Omega'((f, k, d, e, q), \mathcal{R}) = \Omega(q)$ , the bijection of play prefixes immediatly presents a portation of strategies for both players.

Starting positions with non-empty stacks need to be encoded into the MPDS  $P$  first. This can be done by encoding them into the state space in the following sense: When a context would first be initiated on a stack, it first pushes their stack content (The player doing this is unimportant). When the first transition on that stack would be a pop, the player chosing the pop transition will lose, if after the pushes, the symbol to be popped is not on top.  $\blacktriangleleft$

## D Construction of MPDG for the Lower Bound

Formally, we introduce 3 gadgets to prove correctness. Each represents a verification mechanism.

- $G_{comp}^d$ , which checks, whether on top of two stacks is the same encoded word,
- $G_{ind,d}$ , which checks, whether the top of a stack is a valid encoding, and
- $G_{\varphi}^d$ , which checks the two topmost encoded words for the relation  $\sim_{\varphi}$ .

We construct them by induction. Notably, the latter two are constructed by simultaneous induction:  $G_{\varphi}^d$  needs  $G_{ind,d-1}$  and  $G_{\varphi}^{d-1}$  internally. The latter mechanism is described in section 5.4.

### D.1 Construction of $G_{comp}^d$

This gadget expects the top of both stacks to be  $d$ -nested indexings  $w_1, w_2$  of words  $u, v$  of length  $exp_d(len)$ . Eve has a winning strategy, if they index the same word.

To be precise, it is also sufficient if they are not on top, but marked by a delimiter Symbol. In our construction we need the latter case for the verification mechanism  $G_{\varphi}^d$ , where Ana wants to verify the position (of a variable after doubting the symbol is correct) and the valuation is still complete on the second stack. Remembering the correct variable in the control state is no problem as there are a constant amount of variables. For presentational reasons, this differs a little from our detailed version [60].

► **Lemma 21.** *There is a 2-stack MPDG such that Eve has a winning strategy from positions  $(CheckEq_d, w_1\sigma_1\gamma_1\perp, w_2\sigma_2\gamma_2\perp)$  if and only if  $u = v$ . It is constructible in time  $\text{poly}(d + |\Sigma| + n)$ . The maximal number of contexts or phases of any play from such a position is at most  $d + 2$ .*

**Proof idea.** At the top of the stacks are the  $d$ -nested indexings  $w_1$  and  $w_2$ , where  $w_1 = \text{ind}_d(u)$  and  $w_2 = \text{ind}_d(v)$  for some  $u, v \in \Sigma^{\text{max}_d}$ . They have form

$$w_1 = u_0x_0 \dots u_{\text{max}_d}x_{\text{max}_d}, \quad w_2 = v_0x_0 \dots v_{\text{max}_d}x_{\text{max}_d}.$$

Intuitively, Ana removes a sequence of  $(\Sigma\Sigma_{\leq d}^*)^*$  from stack one, until she claims to have found a position  $p$ , where  $u_p \neq v_p$ . She leaves  $x_p$  on top of stack one and store the symbol  $u_p$  in the control state. Then, Eve has to pop a sequence of  $(\Sigma\Sigma_{\leq d}^*)^*$  from stack two. They are supposed to find the corresponding position in  $v$ . Removing the sequence leaves  $v_{p'}x_{p'}$  on top of stack two for some position  $p'$ . After storing  $v_{p'}$  in the control state, Ana may now choose to

1. Believe Eve's choice to be  $p' = p$ . Then, if  $u_p = v_{p'}$ , Eve wins and vice versa.
2. Doubt Eve's choice and claim  $p \neq p'$ . Since  $w_1, w_2$  are  $d$ -nested indexings,  $x_p = \text{ind}_{d-1}(\text{msbf}_d(p))$  and  $x_{p'} = \text{ind}_{d-1}(\text{msbf}_d(p'))$ . Checking  $p \neq p'$  can thus be done by  $(d-1)$ -Equality( $\Sigma_d$ ).

**Number of Contexts and Phases.** The first context or phase starts by Ana removing symbols from stack one. The second context or phase is then started by Eve popping from stack two to find the corresponding position. Then, we can use a copy of  $G_{\text{comp}}^{d-1}$ , where the stacks are swapped, so that it first pops from stack two. This way, the first context or phase of that game merges with the second context or phase. By induction from Lemma 21, this results in a bound of  $1 + (d-1) + 2 = d + 2$  contexts or phases. ◀

## D.2 Construction of $G_{\text{ind}_d}$

This gadget checks the top of the first stack for whether it is a valid  $d$ -nested indexing.

► **Lemma 22.** *There is a 2-stack MPDG  $G_{\text{ind}_d}$  such that Eve has a winning strategy from an initial position (Check  $\text{ind}_d, w\sigma\gamma_1\perp, \gamma_2\perp$ ) if and only if  $w = \text{ind}_d(u)$ , where  $u$  is any word. It is constructible in time  $\text{poly}(d + |\Sigma| + n)$ . Any play has at most  $d + 2$  contexts and  $d + 1$  phases.*

From the initial position, with  $w \in (\Sigma \cup \Sigma_{\leq d}^*)^*$ , the goal is to check whether  $w$  is a valid  $d$ -nested indexing. This holds if and only if the following three conditions are met. (1) The word has the shape  $w = u_0x_0 \dots u_mx_m \in (\Sigma\Sigma_{\leq d}^*)^+$ . (2) Each  $x_p$  is a valid  $(d-1)$ -nested indexing. (3) We have

$$\begin{aligned} x_0 &= \text{ind}_{d-1}(\text{msbf}_d(0)) = \text{ind}_{d-1}(0_d^{\text{exp}_{d-1}(\text{len})}), \\ x_m &= \text{ind}_{d-1}(\text{msbf}_d(\text{max}_d)) = \text{ind}_{d-1}(1_d^{\text{exp}_{d-1}(\text{len})}), \end{aligned}$$

and for all positions  $1 \leq p < m$  with indexing  $x_p = \text{ind}_{d-1}(\text{msbf}_d(i))$  and indexing  $x_{p+1} = \text{ind}_{d-1}(\text{msbf}_d(i'))$  we have  $i' = i + 1$ .

We let Ana choose which condition is violated. In the first case, Eve has to prove that  $w$  is of the form  $(\Sigma\Sigma_{\leq d}^*)^+$ . This can be done by a popping loop.

In the second case, Ana identifies a position  $p$  by removing a sequence from  $\Sigma(\Sigma_{\leq d}^*\Sigma)^*$  and leaving  $x_p$  on top of the stack. We use  $G_{\text{ind}_{d-1}}$  from the induction hypothesis to check whether  $x_p$  is a  $(d-1)$ -nested indexing.

In the last case, there are first-order formulas  $\varphi_0$  and  $\varphi_1$  for the constant conditions and a formula  $\varphi_{+1}$  for the successor relation under most-significant-bit-first encodings. With the induction hypothesis for Lemma 10, we construct the corresponding games  $G_{\varphi_0/\varphi_1/\varphi_{+1}}^{k-1}$ . For checking relation  $\varphi_1$ , before invoking the game, Eve has the task of removing symbols until  $x_m$  is on top of the stack. For checking relation  $\varphi_{+1}$ , Ana first pops symbols to find a position where  $x_p = \text{ind}_{d-1}(\text{msbf}_d(i))$  and  $x_{p+1} = \text{ind}_{d-1}(\text{msbf}_d(i'))$ , but  $i + 1 \neq i'$ .

**Number of Contexts and Phases.** In either case the play starts by popping, leading to a first context or phase on stack one. Actually, in the first case the play already ends after having popped stack one.

In the second case, the play continues to invoke the game  $G_{ind_{d-1}}$ . This adds  $(d-1)+2$  contexts or  $(d-1)+1$  phases respectively, by the induction hypothesis for Lemma 22. However, the first context or phase in  $G_{ind_{d-1}}$  also acts on stack one. So it merges with the previous context or phase for popping from stack one, leading to  $d+1$  contexts or  $d$  phases.

In the last case, the play enters the game  $G_{\varphi}^{d-1}$  for  $\varphi_0$ ,  $\varphi_1$ , or  $\varphi_{+1}$ , leading to  $(d-1)+2$  contexts or  $d-1$  phases respectively, by induction from Lemma 10. With the initial context or phase, we arrive at  $d+2$  contexts and  $d$  phases

Together, this is at most  $d+2$  contexts and  $d$  phases. This covers the required  $d+1$  phases in the Lemma. The base case requires the additional phase.

### D.3 Details on how the players push a valuation

Intuitively, we want to reuse the same principles for pushing successive configuration to push a correctly indexed valuation for variable  $y$ . Eve pushes any sequence and afterwards, Ana can verify that this is indeed a  $(d-1)$ -nested indexing by the use of  $G_{ind_{d-1}}$ . However, when Ana has to choose the valuation, we can not check that Eve pushed the correct position (that is a  $exp_{d-1}(n)$  long sequence from  $\{0_d, 1_d\}$  arbitrarily chosen by Ana). We also cannot swap the roles: Whenever Ana gets the chance of pushing arbitrary long sequences, she can just push symbols infinitely and win the safety winning condition. Thus, we need to let Eve determine when to stop pushing symbols. We do so by letting Eve push sequences in between Ana's choices for single digits of the position. Also, Eve may choose to end the pushing of the sequence at any time. Afterwards, Ana may choose to check, whether the result is a  $(d-1)$ -nested indexing.

### D.4 Adaptions for ordered multi-pushdown systems

It is possible to adapt the lowerbound construction, so that it provides the same strategies for ordered pushdowns. The key idea is to use  $d$  stacks to simulate the  $d$  phases in the lowerbound construction. To this end, we need instances for the gadgets created in the previous sections not only for two stacks, but for combinations of stacks  $j, r$  with  $j < r$ . Further, it should be noted that the gadget  $G_{comp}^d$  can not be used as is, since it pops symbols from both stacks alternatingly, which cannot be done with an ordered pushdown. Instead, it will need some intermediate steps, which will copy the contents to be compared to the another stack (higher in order). Further adaptions are of minor importance and will be mentioned later for completeness.

To this end, we will adapt the gadgets and receive for each stack  $j < r$ ,

- $G_{comp}^d(j, r)$ , comparing the top of stacks  $j$  and  $r$ ,
- $G_{copy}^d(j, r)$ , copying the top of stack indexing from stack  $j$  to  $r$ ,
- $G_{ind_d}(j)$ , checking the top of stack  $j$  for a  $d$ -indexing and
- $G_{\varphi}^d(j)$ , checking the  $\sim_{\varphi}$  relation on the marked indexings on stack  $j$ .

In this,  $G_{copy}^d(j, r)$  will internally use  $G_{comp}^d(j, r)$ , which, in turn, uses  $G_{copy}^{d-1}(j, r)$ . Again, we create the gadget in simultaneous induction together with  $G_{ind_d}(j, r)$ .

The adaptions for  $G_{ind_d}(j)$  and  $G_{\varphi}^d(j)$  are rather small:  $G_{ind_d}(j)$  only needs the stack of the transitions to be changed to  $j$ .  $G_{\varphi}^d(j)$  also needs the stacks to be changed; stack one becomes  $j$  and stack two becomes  $j+1$ . Further, when doubting the suggested variable order, stack  $j$  needs to be emptied, before  $G_{\varphi}^{d-1}(j+1)$  can be called.

## 52:34 On the Complexity of Multi-Pushdown Games

After these adaptations, the construction of the multi-pushdown game simulating an alternating Turing machine is the same as in Section 5. The only difference is, that the ordered pushdown-system created possesses  $d$  stacks.

The following lemma states the same as Lemma 21, but for  $G_{comp}^d(j, r)$ . Let  $u, u' \in \Sigma^{l_d+1}$ ,  $w_1 = ind_d(u)$ ,  $w_2 = ind_d(u')$  and  $j < r \leq n - d$ . Note that the latter requires this gadget to have at least  $d + 2$  stacks.

► **Lemma 23.** *There is an  $n$ -stack ordered MPDG such that Eve has a winning strategy from positions  $(start, [w_1\gamma_1]_j, [w_2\gamma_1]_r)$  if and only if  $u = v$ . It is constructible in time  $\text{poly}(d + |\Sigma| + n)$ .*

► **Base Case** ( $d = 0$ ). The gadget is almost the same as  $G_{comp}^0$ , where transition rules for stack one (2) are swapped for transition rules for stack  $j$  ( $r$ ). The stacks  $j$  to  $r - 1$  are emptied before the transitions popping from  $r$  are executed. Correctness is follows as for  $G_{comp}^0$ .

► **Inductive Case** ( $d > 0$ ). Instantiate  $G_{copy}^{d-1}(j, r + 1)$  on  $(\text{copyPos}, s)$  with out state  $(\text{copyDone}, s)$  for each  $s \in \Sigma$ .

Instantiate  $G_{comp}^{d-1}(r, r + 1)$  on state  $\text{disbelievePos}$ .

Let  $s, s'$  range over  $\Sigma$ .

$(start, (\Sigma\Sigma_{\leq d}^*)^* s, j, (\text{copyPos}, s))$   
 $((\text{copyDone}, s), j, (\text{reproducePos}, s))$   
 $((\text{reproducePos}, s), (\Sigma\Sigma_{\leq d}^*)^* s', r, (\text{claimPos}, s, s'))$   
 $((\text{claimPos}, s, s'), r, (\text{believe}, s, s'))$   
 $((\text{claimPos}, s, s'), r, \text{disbelievePos})$   
 $((\text{believe}, s, s), r, \text{EveWin})$   
 $((\text{believe}, s, s'), r, \text{AnaWin})$   $s \neq s'$

Note that the induction hypothesis (Lemma 24) for the copy gadget holds:  $r + 1 \leq n - d + 1$ . Given by induction (Lemma 24) that each player possesses a strategy from  $((\text{copyPos}, s), [xs\gamma]_j, [x's'\gamma]_r, [\varepsilon]_{r+1})$  to  $((\text{copyDone}, s), [\varepsilon]_j, [x's'\gamma]_r, [xs\gamma]_{r+1})$ , the proof is analogue to the proof for Lemma 21.

And for copying.

► **Lemma 24.** *Let  $u \in \Sigma^{l_d+1}$ ,  $w_1 = ind_d(u)$  and  $j \leq n - d$ . Each player possesses a strategy from  $(start, [w_1\sigma_1\gamma_1]_j, [\varepsilon]_r)$  to  $(Out, [\varepsilon]_j, [w_1]_r)$ .*

*The number of states of this gadget (not counting the states of additionally instantiated gadgets) is polynomially in the size of  $\Sigma$  and  $l_0$ .*

The construction is pretty straightforward: Eve guesses the stackcontent for stack  $r$  and Ana may doubt or believe it.

Instantiate  $G_{ind_d}(r)$  on position  $\text{disbelieveValidity}$ .

Instantiate  $G_{comp}^d(j, r)$  on position  $\text{disbelieveEquality}$ .

$(start, (\Sigma\Sigma_{\leq d}^*)^*, r, \text{pushed})$   
 $(\text{pushed}, r, \text{disbelieveValidity})$   
 $(\text{pushed}, r, \text{disbelieveEquality})$   
 $(\text{pushed}, j, (\Sigma \cup \Sigma_{\leq d})^*, Out)$

Be aware, that the definition of MPDS does not allow for testing a stack for  $\varepsilon$ . One can, however, implement such a transition rule for games given the allowed transition rules.

Now to show that each player possesses a strategy from position  $(start, [w_1\sigma_1\gamma_1]_j, [\varepsilon]_r)$  to  $(start, [\varepsilon]_j, [w_1]_r)$ .

Be aware, that  $r \leq n - d$  holds. The following assumes that Lemma 23 and 22 already hold for the current induction step, which has been shown already.

The strategy for Eve pushes  $w_1$  on stack  $r$  in the first transition. If Ana chooses to go to `disbelieveValidity` or `disbelieveEquality`, Eve wins (Lemma 22 and Lemma 23).

The strategy for Ana analyzes the pushed sequence  $w$  from Eve. If it is not a  $d$ -indexing, Ana wins the play using the move to `disbelieveValidity` (Lemma 22). If it is a valid  $d$ -indexing, but  $w \neq w_1$ , i.e.  $w = ind_d(u')$  with  $u' \neq u$ , Ana wins the play using the move to `disbelieveEquality` (Lemma 23).



# Higher-Order Nonemptiness Step by Step

Paweł Parys 

Institute of Informatics, University of Warsaw, Poland  
parys@mimuw.edu.pl

---

## Abstract

---

We show a new simple algorithm that checks whether a given higher-order grammar generates a nonempty language of trees. The algorithm amounts to a procedure that transforms a grammar of order  $n$  to a grammar of order  $n - 1$ , preserving nonemptiness, and increasing the size only exponentially. After repeating the procedure  $n$  times, we obtain a grammar of order 0, whose nonemptiness can be easily checked. Since the size grows exponentially at each step, the overall complexity is  $n$ -EXPTIME, which is known to be optimal. More precisely, the transformation (and hence the whole algorithm) is linear in the size of the grammar, assuming that the arity of employed nonterminals is bounded by a constant. The same algorithm allows to check whether an infinite tree generated by a higher-order recursion scheme is accepted by an alternating safety (or reachability) automaton, because this question can be reduced to the nonemptiness problem by taking a product of the recursion scheme with the automaton.

A proof of correctness of the algorithm is formalised in the proof assistant Coq. Our transformation is motivated by a similar transformation of Asada and Kobayashi (2020) changing a word grammar of order  $n$  to a tree grammar of order  $n - 1$ . The step-by-step approach can be opposed to previous algorithms solving the nonemptiness problem “in one step”, being compulsorily more complicated.

**2012 ACM Subject Classification** Theory of computation → Rewrite systems

**Keywords and phrases** Higher-order grammars, Nonemptiness, Model-checking, Transformation, Order reduction

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.53

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2009.08174>.

**Supplementary Material** Coq formalisation: <https://github.com/pparys/ho-transform-sbs>

**Funding** Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

## 1 Introduction

Higher-order grammars, also known as higher-order OI grammars [8, 16], generalize context-free grammars: nonterminals of higher-order grammars are allowed to take arguments. Such grammars have been studied actively in recent years, in the context of automated verification of higher-order programs. In this paper we concentrate on a very basic problem of language nonemptiness: is the language generated by a given higher-order grammar nonempty. This problem, being easy for most devices, is not so easy for higher-order grammars. Indeed, it is  $n$ -EXPTIME-complete for grammars of order  $n$  [15].

We give a new simple algorithm solving the language nonemptiness problem. The algorithm amounts to a procedure that transforms a grammar of order  $n$  to a grammar of order  $n - 1$ , preserving nonemptiness, and increasing the size only exponentially. After repeating the procedure  $n$  times, we obtain a grammar of order 0, whose nonemptiness can be easily checked. Since the size grows exponentially at each step, we reach the optimal overall complexity of  $n$ -EXPTIME. In a more detailed view, the complexity looks even better: the size growth is exponential only in the arity of types appearing in the grammar; if the maximal arity is bounded by a constant, the transformation (and hence the whole algorithm) is linear in the size of the grammar.



© Paweł Parys;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 53; pp. 53:1–53:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While a higher-order grammar is a generator of a language of (finite) trees, virtually the same object can be seen as a generator of a single infinite tree (encompassing the whole language). In this context, the grammars are called higher-order recursion schemes. The nonemptiness problem for grammars is easily equivalent to the question whether the tree generated by a given recursion scheme is accepted by a given alternating safety (or reachability) automaton; for the right-to-left reduction, it is enough to product the recursion scheme with the automaton. Thus, our algorithm solves also the latter problem, called a model-checking problem. This problem is decidable and  $n$ -EXPTIME-complete not only for safety or reachability automata, but actually for all parity automata, with multiple proofs using game semantics [17], collapsible pushdown automata [10], intersection types [14], or Krivine machines [20], and with several extensions [5, 3, 6, 21, 18]. The problem for safety automata was tackled in particular by Aehlig [1] and by Kobayashi [12]. To those algorithms we add another one. The main difference between our algorithm and all the others is that we solve the problem step by step, repeatedly reducing the order by one, while most previous algorithms work “in one step”, being compulsorily more complicated. The only proofs that have been reducing the order by one, were proofs using collapsible pushdown automata [10, 3, 6], being very technical (and contained only in unpublished appendices). A reduction of order was also possible for a subclass of recursion schemes, called *safe* recursion schemes [11], but it was not known how to extend it to all recursion schemes.

Comparing the two variants of the model-checking problem for higher-order recursion schemes – involving safety and reachability automata, and involving all parity automata – we have to mention two things. First, while most theoretical results can handle all parity automata, actual tools solving this problem in practice mostly deal only with safety and reachability automata (called also trivial and co-trivial automata) [13, 4, 22, 19]. Second, there exists a polynomial-time (although nontrivial) reduction from the variant involving parity automata to the variant involving safety automata [9].

Our transformation is directly motivated by a recent paper of Asada and Kobayashi [2]. They show how to transform a grammar of order  $n$  generating a language of words to a grammar of order  $n - 1$  generating a language of trees, so that words of the original language are written in leaves of trees of the new language. Unexpectedly, this transformation increases the size of the grammar only polynomially. Our transformation is quite similar, but we start from a grammar generating a language of trees, not words. In effect, on the one hand, we do not say anything specific about the language after the transformation (except that nonemptiness is preserved), and on the other hand, the size growth is exponential, not polynomial.

## 2 Preliminaries

For a number  $k \in \mathbb{N}$  we write  $[k]$  for  $\{1, \dots, k\}$ .

The set of (*simple*) *types* is constructed from a unique ground type  $\mathfrak{o}$  using a binary operation  $\rightarrow$ ; namely  $\mathfrak{o}$  is a type, and if  $\alpha$  and  $\beta$  are types, so is  $\alpha \rightarrow \beta$ . By convention,  $\rightarrow$  associates to the right, that is,  $\alpha \rightarrow \beta \rightarrow \gamma$  is understood as  $\alpha \rightarrow (\beta \rightarrow \gamma)$ . We often abbreviate  $\underbrace{\alpha \rightarrow \dots \rightarrow \alpha}_{\ell} \rightarrow \beta$  as  $\alpha^\ell \rightarrow \beta$ . The *order* of a type  $\alpha$ , denoted  $\text{ord}(\alpha)$ , is defined

by induction:  $\text{ord}(\alpha_1 \rightarrow \dots \rightarrow \alpha_k \rightarrow \mathfrak{o}) = \max(\{0\} \cup \{\text{ord}(\alpha_i) + 1 \mid i \in [k]\})$ ; for example  $\text{ord}(\mathfrak{o}) = 0$ ,  $\text{ord}(\mathfrak{o} \rightarrow \mathfrak{o} \rightarrow \mathfrak{o}) = 1$ , and  $\text{ord}((\mathfrak{o} \rightarrow \mathfrak{o}) \rightarrow \mathfrak{o}) = 2$ .

Having a finite set of typed nonterminals  $\mathcal{X}$ , and a finite set of typed variables  $\mathcal{Y}$ , *terms* over  $(\mathcal{X}, \mathcal{Y})$  are defined by induction:

- every nonterminal  $X \in \mathcal{X}$  of type  $\alpha$  is a term of type  $\alpha$ ;

- every variable  $y \in \mathcal{Y}$  of type  $\alpha$  is a term of type  $\alpha$ ;
- if  $K_1, \dots, K_k$  are terms of type  $\circ$ , then  $\bullet\langle K_1, \dots, K_k \rangle$  and  $\oplus\langle K_1, \dots, K_k \rangle$  are terms of type  $\circ$ ;
- if  $K$  is a term of type  $\alpha \rightarrow \beta$ , and  $L$  is a term of type  $\alpha$ , then  $KL$  is a term of type  $\beta$ .

The type of a term  $K$  is denoted  $\text{tp}(K)$ . The order of a term  $K$ , written  $\text{ord}(K)$ , is defined as the order of its type. We write  $\Omega$  for  $\oplus\langle \rangle$ , and  $\bullet$  for  $\bullet\langle \rangle$ .

The construction  $\oplus\langle K_1, \dots, K_k \rangle$  is an alternative; such a term reduces to one of the terms  $K_1, \dots, K_k$ . This construction is used to introduce nondeterminism to grammars (defined below). In the special case of  $k = 0$  (when we write  $\Omega$ ) no reduction is possible; thus  $\Omega$  denotes divergence.

The construction  $\bullet\langle K_1, \dots, K_k \rangle$  can be seen as a generator of a tree node with  $k$  children; subtrees starting in these children are described by the terms  $K_1, \dots, K_k$ . In a usual presentation, nodes are labeled by letters from some finite alphabet. In this paper, however, we do not care about the exact letters contained in generated trees, only about language nonemptiness, hence we do not write these letters at all (in other words, we use a single-letter alphabet, where  $\bullet$  is the only letter). Actually, in the sequel we even do not consider trees; we rather say that  $\bullet\langle K_1, \dots, K_k \rangle$  is convergent if all  $K_1, \dots, K_k$  are convergent (which can be rephrased as: the language generated from  $\bullet\langle K_1, \dots, K_k \rangle$  is nonempty if the languages generated from all  $K_1, \dots, K_k$  are nonempty).

A (*higher-order*) *grammar* is a tuple  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ , where  $\mathcal{X}$  a finite set of typed nonterminals,  $X_0 \in \mathcal{X}$  is a starting nonterminal of type  $\circ$ , and  $\mathcal{R}$  a function assigning to every nonterminal  $X \in \mathcal{X}$  a rule of the form  $X y_1 \dots y_k \rightarrow R$ , where  $\text{tp}(X) = (\text{tp}(y_1) \rightarrow \dots \rightarrow \text{tp}(y_k) \rightarrow \circ)$ , and  $R$  is a term of type  $\circ$  over  $(\mathcal{X}, \{y_1, \dots, y_k\})$ . The order of a grammar is defined as the maximum of orders of its nonterminals.

Having a grammar  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ , for every set of variables  $\mathcal{Y}$  we define a *reduction relation*  $\longrightarrow_{\mathcal{G}}$  between terms over  $(\mathcal{X}, \mathcal{Y})$  and sets of such terms, as the least relation such that

- (1)  $X K_1 \dots K_k \longrightarrow_{\mathcal{G}} \{R[K_1/y_1, \dots, K_k/y_k]\}$  if the rule for  $X$  is  $X y_1 \dots y_k \rightarrow R$ , where  $R[K_1/y_1, \dots, K_k/y_k]$  denotes the term obtained from  $R$  by substituting  $K_i$  for  $y_i$  for all  $i \in [k]$ ,
- (2)  $\bullet\langle K_1, \dots, K_k \rangle \longrightarrow_{\mathcal{G}} \{K_1, \dots, K_k\}$ , and
- (3)  $\oplus\langle K_1, \dots, K_k \rangle \longrightarrow_{\mathcal{G}} \{K_i\}$  for every  $i \in [k]$ .

We say that a term  $M$  is  $\mathcal{G}$ -convergent if  $M \longrightarrow_{\mathcal{G}} \mathcal{N}$  for some set  $\mathcal{N}$  of  $\mathcal{G}$ -convergent terms. This is an inductive definition; in particular, the base case is when  $M \longrightarrow_{\mathcal{G}} \emptyset$ . In other words,  $M$  is  $\mathcal{G}$ -convergent if there is a finite tree labeled by terms where for each node, the node and its children satisfy one of (1)-(3). Moreover, the grammar  $\mathcal{G}$  is *convergent* if its starting nonterminal  $X_0$  is  $\mathcal{G}$ -convergent.

### 3 Transformation

In this section we present a transformation, called *order-reducing transformation*, resulting in the main theorem of this paper:

► **Theorem 3.1.** *For any  $n \geq 1$ , there exists a transformation from order- $n$  grammars to order- $(n - 1)$  grammars, and a polynomial  $p_n$  such that, for any order- $n$  grammar  $\mathcal{G}$ , the resulting grammar  $\mathcal{G}^\dagger$  is convergent if and only if  $\mathcal{G}$  is convergent, and  $|\mathcal{G}^\dagger| \leq 2^{p_n(|\mathcal{G}|)}$ .*

## 53:4 Higher-Order Nonemptiness Step by Step

**Intuitions.** Let us first present intuitions behind our transformation. While reducing the order, we have to replace, in particular, order-1 functions by order-0 terms. Consider for example a term  $KL$  of type  $\circ$ , where  $K$  has type  $\circ \rightarrow \circ$ . Notice that  $L$  generates trees that are inserted somewhere in contexts generated by  $K$ . Thus, when is  $KL$  convergent? There are two possibilities. First, maybe  $K$  is convergent without using its argument at all. Second, maybe  $K$  can be convergent but only using its argument, and then  $L$  also has to be convergent. Notice that in the first case  $K\Omega$  is convergent (i.e.,  $K$  is convergent even if the argument is not convergent), and in the second case  $K\bullet$  is convergent (i.e.,  $K$  is convergent if its argument is convergent). In the transformation, we transform  $K$  into two order-0 terms,  $K_0$  and  $K_1$  corresponding to  $K\Omega$  and  $K\bullet$ , and then we replace  $KL$  by  $\oplus\langle K_0, \bullet\langle K_1, L \rangle \rangle$ .

As a full example, consider an order-1 grammar with the following rules:

$$X \rightarrow YZ, \quad Yx \rightarrow \oplus\langle \bullet, x \rangle, \quad Z \rightarrow \bullet.$$

It will be transformed to the order-0 grammar with the following rules:

$$X \rightarrow \oplus\langle Y_0, \bullet\langle Y_1, Z \rangle \rangle, \quad Y_0 \rightarrow \oplus\langle \bullet, \Omega \rangle, \quad Y_1 \rightarrow \oplus\langle \bullet, \bullet \rangle, \quad Z \rightarrow \bullet.$$

Notice that the original grammar is convergent “for two reasons”: the  $\oplus$  node in the rule for  $Y$  may reduce either to the first possibility (i.e., to  $\bullet$ ), or to the second possibility (i.e., to  $x$ ), in which case convergence follows from convergence of the argument  $Z$ . This is reflected by the two possibilities available for the  $\oplus$  node in the new rule for  $X$ : we either choose the first possibility and we depend only on convergence of  $Y_0$ , or we choose the the second possibility and we depend on convergence of both  $Y_1$  and  $Z$ . Notice that after replacing the (old and new) rule for  $Z$  by  $Z \rightarrow \Omega$ , the modified grammars remain convergent thanks to the first possibility above. Likewise, after replacing the original rule for  $Y$  by  $Yx \rightarrow x$ , the new rules will be  $Y_0 \rightarrow \Omega$  and  $Y_1 \rightarrow \bullet$ , and the modified grammars remain convergent thanks to the second possibility above. However, after applying both these replacements simultaneously, the grammars stop to be convergent.

If our term  $K$  takes multiple order-0 arguments, say we have  $KL_1 \dots L_k$ , while transforming  $K$  we need  $2^k$  variants of the term: each of the arguments may be either used (replaced by  $\bullet$ ) or not used (replaced by  $\Omega$ ). This is why we have the exponential blow-up. Let us compare this quickly with the transformation of Asada and Kobayashi [2], which worked for grammars generating words (i.e., trees where every node has at most one child). In their case, at most one of the arguments  $L_i$  could be used, so they needed only  $k + 1$  variants of  $K$ ; this is why their transformation was polynomial.

For higher-order grammars we apply the same idea: functions of order 1 are replaced by terms of order 0, and then the order of any higher-order function drops down by one. For example, consider a grammar with the following rules:

$$X \rightarrow TY, \quad Ty \rightarrow y(y\bullet), \quad Yx \rightarrow \oplus\langle \bullet, x \rangle.$$

The nonterminal  $Y$  is again of type  $\circ \rightarrow \circ$ , hence it is replaced by two nonterminals  $Y_0, Y_1$  of type  $\circ$ , describing the situation when the parameter  $x$  is either not used or used. Likewise, the corresponding parameter  $y$  of  $T$  is replaced by two parameters  $y_0, y_1$ . The resulting grammar will have the following rules:

$$X \rightarrow TY_0Y_1, \quad Ty_0y_1 \rightarrow \oplus\langle y_0, \bullet\langle y_1, \oplus\langle y_0, \bullet\langle y_1, \bullet \rangle \rangle \rangle \rangle, \quad Y_0 \rightarrow \oplus\langle \bullet, \Omega \rangle, \quad Y_1 \rightarrow \oplus\langle \bullet, \bullet \rangle.$$

**Formal definition.** We now formalize the above intuitions. Having a type, we are interested in cutting off its suffix being of order 1. Thus, we use the notation  $\alpha_1 \rightarrow \dots \rightarrow \alpha_k \Rightarrow \circ^\ell \rightarrow \circ$  for a type  $\alpha_1 \rightarrow \dots \rightarrow \alpha_k \rightarrow \circ^\ell \rightarrow \circ$  such that either  $k = 0$  or  $\alpha_k \neq \circ$ . Notice that every

type  $\alpha$  can be uniquely represented in this form. We remark that some among the types  $\alpha_1, \dots, \alpha_{k-1}$  (but not  $\alpha_k$ ) may be  $\mathfrak{o}$ . For a type  $\alpha$  we write  $\text{gar}(\alpha)$  (“ground arity”) for the number  $\ell$  for which we can write  $\alpha = (\alpha_1 \rightarrow \dots \rightarrow \alpha_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})$ ; we also extend this to terms:  $\text{gar}(M) = \text{gar}(\text{tp}(M))$ .

We transform terms of type  $\alpha$  to terms of type  $\alpha^\dagger$ , which is defined by induction:

$$(\alpha_1 \rightarrow \dots \rightarrow \alpha_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})^\dagger = \left( (\alpha_1^\dagger)^{2^{\text{gar}(\alpha_1)}} \rightarrow \dots \rightarrow (\alpha_k^\dagger)^{2^{\text{gar}(\alpha_k)}} \rightarrow \mathfrak{o} \right).$$

Thus, we remove all trailing order-0 arguments, and we multiply (and recursively transform) remaining arguments.

For a finite set  $S$ , we write  $2^S$  for the set of functions  $A: S \rightarrow \{0, 1\}$ . Moreover, we assume some fixed order on functions in  $2^S$ , and we write  $P(Q_A)_{A \in 2^S}$  for an application  $P Q_{A_1} \dots Q_{A_{2^{|S|}}}$ , where  $A_1, \dots, A_{2^{|S|}}$  are all the function from  $2^S$  listed in the fixed order. The only function in  $2^\emptyset$  is denoted  $\emptyset$ .

Fix a grammar  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ . For every nonterminal  $X$  and for every function  $A \in 2^{[\text{gar}(X)]}$  we consider a nonterminal  $X_A^\dagger$  of type  $(\text{tp}(X))^\dagger$ . As the new set of nonterminals we take  $\mathcal{X}^\dagger = \{X_A^\dagger \mid X \in \mathcal{X}, A \in 2^{[\text{gar}(X)]}\}$ . Likewise, for every variable  $y$  and for every function  $A \in 2^{[\text{gar}(y)]}$  we consider a variable  $y_A^\dagger$  of type  $(\text{tp}(y))^\dagger$ , and for a set of variables  $\mathcal{Y}$  we denote  $\mathcal{Y}^\dagger = \{y_A^\dagger \mid y \in \mathcal{Y}, A \in 2^{[\text{gar}(y)]}\}$ .

We now define a function  $\text{tr}$  transforming terms. Its value  $\text{tr}(A, Z, M)$  is defined when  $M$  is a term over some  $(\mathcal{X}, \mathcal{Y})$ , and  $A \in 2^{[\text{gar}(M)]}$ , and  $Z: \mathcal{Y} \rightarrow \{0, 1\}$  is a partial function such that  $\text{dom}(Z)$  contains only variables of type  $\mathfrak{o}$ . The intention is that  $A$  specifies which among trailing order-0 arguments can be used, and  $Z$  specifies which order-0 variables (among those in  $\text{dom}(Z)$ ) can be used. The transformation is defined by induction on the structure of  $M$ , as follows:

- (1)  $\text{tr}(A, Z, X) = X_A$  for  $X \in \mathcal{X}$ ;
- (2)  $\text{tr}(A, Z, y) = y_A$  for  $y \in \mathcal{Y} \setminus \text{dom}(Z)$ ;
- (3)  $\text{tr}(A, Z, z) = \Omega$  if  $Z(z) = 0$ ;
- (4)  $\text{tr}(A, Z, z) = \bullet$  if  $Z(z) = 1$ ;
- (5)  $\text{tr}(\emptyset, Z, \bullet \langle K_1, \dots, K_k \rangle) = \bullet \langle \text{tr}(\emptyset, Z, K_1), \dots, \text{tr}(\emptyset, Z, K_k) \rangle$ ;
- (6)  $\text{tr}(\emptyset, Z, \oplus \langle K_1, \dots, K_k \rangle) = \oplus \langle \text{tr}(\emptyset, Z, K_1), \dots, \text{tr}(\emptyset, Z, K_k) \rangle$ ;
- (7)  $\text{tr}(A, Z, K L) = \oplus \langle \text{tr}(A[\ell+1 \mapsto 0], Z, K), \bullet \langle \text{tr}(A[\ell+1 \mapsto 1], Z, K), \text{tr}(\emptyset, Z, L) \rangle \rangle$  if  $\text{tp}(K) = (\mathfrak{o}^{\ell+1} \rightarrow \mathfrak{o})$ ;
- (8)  $\text{tr}(A, Z, K L) = (\text{tr}(A, Z, K)) (\text{tr}(B, Z, L))_{B \in 2^{[\text{gar}(L)]}}$  if  $\text{tp}(K) = (\alpha_1 \rightarrow \dots \rightarrow \alpha_k \Rightarrow \mathfrak{o}^\ell \rightarrow \mathfrak{o})$  with  $k \geq 1$ .

For every rule  $X y_1 \dots y_k z_1 \dots z_\ell \rightarrow R$  in  $\mathcal{R}$ , where  $\ell = \text{gar}(X)$ , and for every function  $A \in 2^{[\ell]}$ , to  $\mathcal{R}^\dagger$  we take the rule

$$X_A^\dagger (y_{1,B}^\dagger)_{B \in 2^{[\text{gar}(y_1)]}} \dots (y_{k,B}^\dagger)_{B \in 2^{[\text{gar}(y_k)]}} \rightarrow \text{tr}(\emptyset, [z_i \mapsto A(\ell+1-i) \mid i \in [\ell]], R).$$

In the function  $A$  it is more convenient to count arguments from right to left (then we do not need to shift the domain in Case (7) above), but it is more natural to have variables  $z_1, \dots, z_\ell$  numbered from left to right; this is why in the rule for  $X_A^\dagger$  we assign to  $z_i$  the value  $A(\ell+1-i)$ , not  $A(i)$ .

Finally, the resulting grammar  $\mathcal{G}^\dagger$  is  $(\mathcal{X}^\dagger, X_{0,\emptyset}^\dagger, \mathcal{R}^\dagger)$ .

## 4 Complexity

In this section we analyze complexity of our transformation. First, we formally define the *size* of a grammar. The size of a term is defined by induction on its structure:

$$\begin{aligned} |X| = |y| &= 1, & |K L| &= 1 + |K| + |L|, \\ |\bullet\langle K_1, \dots, K_k \rangle| &= |\oplus\langle K_1, \dots, K_k \rangle| = 1 + |K_1| + \dots + |K_k|. \end{aligned}$$

Then  $|\mathcal{G}|$ , the size of  $\mathcal{G}$  is defined as the sum of  $|R| + k$  over all rules  $X y_1 \dots y_k \rightarrow R$  of  $\mathcal{G}$ . In Asada and Kobayashi [2] such a size is called *Curry-style size*; it does not include sizes of types of employed variables.

We say that a type  $\alpha$  is a *subtype* of a type  $\beta$  if either  $\alpha = \beta$ , or  $\beta = (\beta_1 \rightarrow \beta_2)$  and  $\alpha$  is a subtype of  $\beta_1$  or of  $\beta_2$ . We write  $A_{\mathcal{G}}$  for the largest arity of subtypes of types of nonterminals in a grammar  $\mathcal{G}$ . Notice that types of other objects appearing in  $\mathcal{G}$ , namely variables and subterms of right sides of rules, are subtypes of types of nonterminals, hence their arity is also bounded by  $A_{\mathcal{G}}$ . It is reasonable to consider large grammars, consisting of many rules, where simultaneously the maximal arity  $A_{\mathcal{G}}$  is respectively small.

While the exponential bound mentioned in Theorem 3.1 is obtained by applying the order-reducing transformation to an arbitrary grammar, the complexity becomes slightly better if we first apply a preprocessing step. This is in particular necessary, if we want to obtain linear dependence in the size of  $\mathcal{G}$  (and exponential only in the maximal arity  $A_{\mathcal{G}}$ ). The preprocessing, making sure that the grammar is in a *simple form* (defined below) amounts to splitting large rules into multiple smaller rules. A similar preprocessing is present already in prior work [13, 2, 7], however our definition of a simple form is slightly more liberal, so that the order reduction applied to a grammar in a normal form gives again a grammar in a normal form.

An *application depth* of a term  $R$  is defined as the maximal number of applications on a single branch in  $R$ , where a compound application  $K L_1 \dots L_k$  counts only once. More formally, we define by induction:

$$\begin{aligned} \text{ad}(\bullet\langle K_1, \dots, K_k \rangle) &= \text{ad}(\oplus\langle K_1, \dots, K_k \rangle) = \max\{\text{ad}(K_i) \mid i \in [k]\}, \\ \text{ad}(X K_1 \dots K_k) &= \text{ad}(y K_1 \dots K_k) = \max(\{0\} \cup \{\text{ad}(K_i) + 1 \mid i \in [k]\}). \end{aligned}$$

We say that a grammar  $\mathcal{G}$  is in a *simple form* if the right side of each its rule has application depth at most 2.

Any grammar  $\mathcal{G}$  can be converted to a grammar in a simple form, as follows. Consider a rule  $X y_1 \dots y_k \rightarrow R$ , and a subterm of  $R$  of the form  $f K_1 \dots K_m$ , where  $f$  is a nonterminal or a variable, but some  $K_i$  already has application depth 2. Then we replace the occurrence of  $K_i$  with  $Y y_1 \dots y_k$  (being a term of application depth 1) for a fresh nonterminal  $Y$ , and we add the rule  $Y y_1 \dots y_k x_1 \dots x_s \rightarrow K_i x_1 \dots x_s$  (whose right side already has application depth 2; the additional variables  $x_1, \dots, x_s$  are added to ensure that the type is  $\circ$ ). By repeating such a replacement for every “bad” subterm of every rule, we clearly obtain a grammar in a simple form.

► **Lemma 4.1.** *Let  $\mathcal{G}'$  be the grammar in a simple form obtained by the above simplification procedure from a grammar  $\mathcal{G}$ . Then  $\text{ord}(\mathcal{G}') = \text{ord}(\mathcal{G})$ , and  $A_{\mathcal{G}'} \leq 2A_{\mathcal{G}}$ , and  $|\mathcal{G}'| = \mathcal{O}(A_{\mathcal{G}} \cdot |\mathcal{G}|)$ . The procedure can be performed in time linear in its output size.*

**Proof.** The parts about the order and about the running time are obvious.

Types of nonterminals originating from  $\mathcal{G}$  remain unchanged. The type of a fresh nonterminal  $Y$  introduced in the procedure is of the form  $\alpha_1 \rightarrow \dots \rightarrow \alpha_k \rightarrow \beta_1 \rightarrow \dots \rightarrow \beta_s \rightarrow \circ$ ,

where all  $\alpha_i$  and  $\beta_i$  are types present also in  $\mathcal{G}$ . The arity of the whole type is  $k + s$ , where  $k$  is the arity of the original nonterminal  $X$  (hence it is bounded by  $A_{\mathcal{G}}$ ), and  $s$  is bounded by the arity of the type of  $K_i$  (hence also by  $A_{\mathcal{G}}$ ).

In order to bound the size of the resulting grammar, notice that the considered replacement is performed at most once for every subterm of the right side of every rule, hence the number of replacements is bounded by  $|\mathcal{G}|$ . Each such a replacement increases the size of the grammar by at most  $\mathcal{O}(A_{\mathcal{G}})$ . ◀

► **Lemma 4.2.** *For every grammar  $\mathcal{G}$  in a simple form, the grammar  $\mathcal{G}^\dagger$  (i.e., the result of the order-reducing transformation) is also in a simple form, and  $\text{ord}(\mathcal{G}^\dagger) = \max(0, \text{ord}(\mathcal{G}) - 1)$ , and  $A_{\mathcal{G}^\dagger} \leq A_{\mathcal{G}} \cdot 2^{A_{\mathcal{G}}}$ , and  $|\mathcal{G}^\dagger| = \mathcal{O}(|\mathcal{G}| \cdot 2^{5 \cdot A_{\mathcal{G}}})$ . Moreover, the transformation can be performed in time linear in its output size.*

**Proof.** The part about the running time is obvious. It is also easy to see by induction that  $\text{ord}(\alpha^\dagger) = \max(0, \text{ord}(\alpha) - 1)$ . It follows that the order of the grammar satisfies the same equality, because nonterminals of  $\mathcal{G}^\dagger$  have type  $\alpha^\dagger$  for  $\alpha$  being the type of a corresponding nonterminal of  $\mathcal{G}$ .

Recall that in the type  $\alpha^\dagger$  obtained from  $\alpha = (\alpha_1 \rightarrow \dots \rightarrow \alpha_k \rightarrow \circ)$ , every  $\alpha_i$  either disappears or becomes (transformed and) repeated  $2^{\text{gar}(\alpha_i)}$  times, that is, at most  $2^{A_{\mathcal{G}}}$  times. This implies the inequality concerning  $A_{\mathcal{G}^\dagger}$ .

Every compound application can be written as  $f K_1 \dots K_k L_1 \dots L_\ell$ , where  $f$  is a nonterminal or a variable, and  $\ell = \text{gar}(f)$ . In such a term, every  $K_i$  (after transforming) becomes repeated  $2^{\text{gar}(K_i)}$  times, that is, at most  $2^{A_{\mathcal{G}}}$  times. Then, for every  $L_i$  we duplicate the outcome and we append a small prefix; this duplication happens  $\ell$  times, that is, at most  $A_{\mathcal{G}}$  times. In consequence, we easily see by induction that while transforming a term of application depth  $d$ , its size gets multiplied by at most  $\mathcal{O}(2^{2^d \cdot A_{\mathcal{G}}})$ . Moreover, every nonterminal  $X$  is repeated  $2^{\text{gar}(X)}$  times, that is, at most  $2^{A_{\mathcal{G}}}$  times. Because the application depth of right sides of rules is at most 2, this bounds the size of the new grammar by  $\mathcal{O}(|\mathcal{G}| \cdot 2^{5 \cdot A_{\mathcal{G}}})$ .

Looking again at the above description of the transformation, we can notice that the application depth cannot grow; in consequence the property of being in a simple form is preserved. ◀

Thus, if we want to check nonemptiness of a grammar  $\mathcal{G}$  of order  $n$ , we can first convert it to a simple form, and then apply the order-reducing transformation  $n$  times. This gives us a grammar of order 0, whose nonemptiness can be checked in linear time. By Lemmata 4.1 and 4.2, the whole algorithm works in time  $n$ -fold exponential in  $A_{\mathcal{G}}$  and linear in  $|\mathcal{G}|$ .

If the original grammar  $\mathcal{G}$  generates a language of words, we can start by applying the polynomial-time transformation of Asada and Kobayashi [2], which converts  $\mathcal{G}$  into an equivalent grammar of order  $n - 1$  (generating a language of trees); then we can continue as above. Because their transformation is also linear in  $|\mathcal{G}|$ , and increases the arity only quadratically, in this case we obtain an algorithm working in time  $(n - 1)$ -fold exponential in  $A_{\mathcal{G}}$  and linear in  $|\mathcal{G}|$ .

## 5 Correctness

In this section we finish a proof of Theorem 3.1 by showing that the grammar  $\mathcal{G}^\dagger$  resulting from transforming a grammar  $\mathcal{G}$  is convergent if and only if the original grammar  $\mathcal{G}$  is convergent. This proof is also formalised in the proof assistant Coq, and available at GitHub (<https://github.com/pparys/ho-transform-sbs>). The strategy of our proof is similar as in

Asada and Kobayashi [2]. Namely, we first show that reductions performed by  $\mathcal{G}$  can be reordered, so that we can postpone substituting for (trailing) variables of order 0. To store such postponed substitutions, called *explicit substitutions*, we introduce *extended terms*. Then, we show that such reordered reductions in  $\mathcal{G}$  are in a direct correspondence with reductions in  $\mathcal{G}^\dagger$ .<sup>1</sup>

**Extended terms.** In the sequel, terms defined previously are sometimes called non-extended terms, in order to distinguish them from extended terms defined below. Having a finite set of typed nonterminals  $\mathcal{X}$ , and a finite set  $\mathcal{Z}$  of variables of type  $\mathfrak{o}$ , *extended terms* over  $(\mathcal{X}, \mathcal{Z})$  are defined by induction:

- if  $z \notin \mathcal{Z}$  is a variable of type  $\mathfrak{o}$ , and  $E$  is an extended term over  $(\mathcal{X}, \mathcal{Z} \uplus \{z\})$ , and  $L$  is a non-extended term of type  $\mathfrak{o}$  over  $(\mathcal{X}, \mathcal{Z})$ , then  $E\langle L/z \rangle$  is an extended term over  $(\mathcal{X}, \mathcal{Z})$ ;
- every non-extended term of type  $\mathfrak{o}$  over  $(\mathcal{X}, \mathcal{Z})$  is an extended term over  $(\mathcal{X}, \mathcal{Z})$ .

The construction of the form  $E\langle L/z \rangle$  is called an *explicit substitution*. Intuitively, it denotes the term obtained by substituting  $L$  for  $z$  in  $E$ . Notice that the variable  $z$  being free in  $E$  becomes bound in  $E\langle L/z \rangle$ , and that explicit substitutions are allowed only for the ground type  $\mathfrak{o}$ .

Of course a (non-extended or extended) term over  $(\mathcal{X}, \mathcal{Z})$  can be also seen as a term over  $(\mathcal{X}, \mathcal{Z}')$ , where  $\mathcal{Z}' \supseteq \mathcal{Z}$ . In the sequel, such extending of the set of variables is often performed implicitly.

Having a grammar  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ , for every set  $\mathcal{Z}$  of variables of type  $\mathfrak{o}$  we define an *ext-reduction* relation  $\rightsquigarrow_{\mathcal{G}}$  between extended terms over  $(\mathcal{X}, \mathcal{Z})$  and sets of such terms, as the least relation such that

- (1)  $X K_1 \dots K_k L_1 \dots L_\ell \rightsquigarrow_{\mathcal{G}} \{R[K_1/y_1, \dots, K_k/y_k, z'_1/z_1, \dots, z'_\ell/z_\ell]\langle L_1/z'_1 \rangle \dots \langle L_\ell/z'_\ell \rangle\}$  if  $\ell = \text{gar}(X)$ , and  $\mathcal{R}(X) = (X y_1 \dots y_k z_1 \dots z_\ell \rightarrow R)$ , and  $z'_1, \dots, z'_\ell$  are fresh variables of type  $\mathfrak{o}$  not appearing in  $\mathcal{Z}$ ,
- (2)  $\bullet\langle K_1, \dots, K_k \rangle \rightsquigarrow_{\mathcal{G}} \{K_1, \dots, K_k\}$ ,
- (3)  $\oplus\langle K_1, \dots, K_k \rangle \rightsquigarrow_{\mathcal{G}} \{K_i\}$  for every  $i \in [k]$ ,
- (4)  $z\langle L/z \rangle \rightsquigarrow_{\mathcal{G}} \{L\}$ ,
- (5)  $z'\langle L/z \rangle \rightsquigarrow_{\mathcal{G}} \{z'\}$  if  $z' \neq z$ , and
- (6)  $E\langle L/z \rangle \rightsquigarrow_{\mathcal{G}} \{F\langle L/z \rangle \mid F \in \mathcal{F}\}$  whenever  $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ .

We say that an extended term  $E$  over  $(\mathcal{X}, \emptyset)$  is  $\mathcal{G}$ -*ext-convergent* if  $E \rightarrow_{\mathcal{G}} \mathcal{F}$  for some set  $\mathcal{F}$  of  $\mathcal{G}$ -ext-convergent extended terms. The grammar  $\mathcal{G}$  is *ext-convergent* if its starting nonterminal  $X_0$  is  $\mathcal{G}$ -ext-convergent.

There is an “expand” function from extended terms to non-extended terms, which performs all the explicit substitutions written in front of an extended term:

$$\text{exp}(K\langle L_1/z_1 \rangle \dots \langle L_\ell/z_\ell \rangle) = K[L_1/z_1] \dots [L_\ell/z_\ell].$$

We also write  $\text{exp}(\mathcal{F})$  for  $\{\text{exp}(F) \mid F \in \mathcal{F}\}$  (where  $\mathcal{F}$  is a set of extended terms). The following lemma, saying that we can consider ext-convergence instead of convergence, can be proved in a standard way (actually, Asada and Kobayashi have a very similar lemma [2, Lemma 18]); a proof can be found in the full version of the paper (Appendix A).

<sup>1</sup> Asada and Kobayashi have an additional step in their proof, namely a reduction to the case of recursion-free grammars. This step turns out to be redundant, at least in the case of our transformation.

► **Lemma 5.1.** *Let  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$  be a grammar. An extended term  $E$  over  $(\mathcal{X}, \emptyset)$  is  $\mathcal{G}$ -ext-convergent if and only if  $\text{exp}(E)$  is  $\mathcal{G}$ -convergent. In particular  $\mathcal{G}$  is ext-convergent if and only if it is convergent.*

We extend the transformation function to extended terms, by adding the following rule, where  $E\langle L/z \rangle$  is an extended term over  $(\mathcal{X}, \mathcal{Z})$ , and  $Z \in 2^{\mathcal{Z}}$  (the first argument is always  $\emptyset$ , because all extended terms are of type  $\circ$ ):

$$(9) \text{tr}(\emptyset, Z, E\langle L/z \rangle) = \oplus \langle \text{tr}(\emptyset, Z[z \mapsto 0], E), \bullet \langle \text{tr}(\emptyset, Z[z \mapsto 1], E), \text{tr}(\emptyset, Z, L) \rangle \rangle.$$

**Between ext-convergence and convergence of  $\mathcal{G}^\dagger$ .** Once we know that convergence and ext-convergence of  $\mathcal{G}$  are equivalent (cf. Lemma 5.1), it remains to prove that ext-convergence of  $\mathcal{G}$  is equivalent to convergence of  $\mathcal{G}^\dagger$ , which is the subject of Lemma 5.2:

► **Lemma 5.2.** *Let  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$  be a grammar. An extended term  $E$  over  $(\mathcal{X}, \emptyset)$  is  $\mathcal{G}$ -ext-convergent if and only if  $\text{tr}(\emptyset, \emptyset, E)$  is  $\mathcal{G}^\dagger$ -convergent. In particular  $\mathcal{G}$  is ext-convergent if and only if  $\mathcal{G}^\dagger$  is convergent.*

The remaining part of this section is devoted to a proof of this lemma. Fix a grammar  $\mathcal{G} = (\mathcal{X}, X_0, \mathcal{R})$ . Of course the second part (concerning the grammars) follows from the first part (concerning an extended term) applied to the starting nonterminal  $X_0$ . It is thus enough to prove the first part. We start with the left-to-right direction (i.e., from  $\mathcal{G}$ -ext-convergence of  $E$  to  $\mathcal{G}^\dagger$ -convergence of  $\text{tr}(\emptyset, \emptyset, E)$ ). We need two simple auxiliary lemmata. The first of them says that the  $\text{tr}$  function commutes with substitution:

► **Lemma 5.3.** *Let  $R[K_1/y_1, \dots, K_k/y_k]$  be a term over  $(\mathcal{X}, \mathcal{Z})$ , let  $A \in 2^{\text{gar}(R)}$ , and let  $Z \in 2^{\mathcal{Z}}$ . Then*

$$\text{tr}(A, Z, R[K_1/y_1, \dots, K_k/y_k]) = (\text{tr}(A, Z, R))[\text{tr}(B, Z, K_i)/y_{i,B}^\dagger \mid i \in [k], B \in 2^{\text{gar}(K_i)}].$$

**Proof.** A straightforward induction on the structure of  $R$ . ◀

The second lemma says that by increasing values of the function  $Z$  we can make the transformed term only more convergent:

► **Lemma 5.4.** *Let  $E$  be an extended term over  $(\mathcal{X}, \mathcal{Z} \uplus \{z\})$ , and let  $Z \in 2^{\mathcal{Z}}$ . If  $\text{tr}(\emptyset, Z[z \mapsto 0], E)$  is  $\mathcal{G}^\dagger$ -convergent, then also  $\text{tr}(\emptyset, Z[z \mapsto 1], E)$  is  $\mathcal{G}^\dagger$ -convergent.*

**Proof.** Denote  $P^0 = \text{tr}(\emptyset, Z[z \mapsto 0], E)$  and  $P^1 = \text{tr}(\emptyset, Z[z \mapsto 1], E)$ . Tracing the rules of the transformation function, we can see that  $P^0$  and  $P^1$  are created in the same way, with the exception that occurrences of  $z$  in  $E$  are transformed to  $\Omega$  in  $P^0$ , and to  $\bullet$  in  $P^1$ . Thus,  $P^1$  can be obtained from  $P^0$  by replacing some occurrences of  $\Omega$  to  $\bullet$ . We know that  $P^0$  is  $\mathcal{G}^\dagger$ -convergent, which means that it can be rewritten using the  $\rightarrow_{\mathcal{G}}$  relation until reaching empty sets. Moreover, the subterms  $\Omega$  (which are present in  $P^0$ , but not in  $P^1$ ) cannot be reached during this rewriting, because  $\Omega$  is not  $\mathcal{G}^\dagger$ -convergent. Thus,  $P^1$  can be rewritten in exactly the same way as  $P^0$ , so it is also  $\mathcal{G}^\dagger$ -convergent. ◀

The next lemma shows how ext-reductions of  $\mathcal{G}$  are reflected in  $\mathcal{G}^\dagger$ :

► **Lemma 5.5.** *Let  $E$  be an extended term over  $(\mathcal{X}, \mathcal{Z})$  and let  $Z \in 2^{\mathcal{Z}}$ . If  $E \rightsquigarrow_{\mathcal{G}} F$  and  $\text{tr}(\emptyset, Z, F)$  is  $\mathcal{G}^\dagger$ -convergent for every  $F \in \mathcal{F}$ , then  $\text{tr}(\emptyset, Z, E)$  is also  $\mathcal{G}^\dagger$ -convergent.*

### 53:10 Higher-Order Nonemptiness Step by Step

**Proof.** Induction on the definition of  $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$ . We analyze particular cases appearing in the definition. Missing details are given in the full version of the paper (Appendix B).

In Case (1)  $E$  consists of an application of arguments to some nonterminal  $X$ . For simplicity of presentation, suppose that  $X$  has two arguments:  $y$  of positive order, and  $z$  of order 0 (the general case is handled in the full version of the paper). Then

$$E = X K L, \quad \text{and} \quad \mathcal{F} = \{F\} \quad \text{for} \quad F = R[K/y, z'/z]\langle L/z' \rangle,$$

where  $\mathcal{R}(X) = (X y z \rightarrow R)$  and  $z'$  is a fresh variable of type  $\circ$  not appearing in  $\mathcal{Z}$ . For  $j \in \{0, 1\}$  let

$$P^j = \text{tr}([1 \mapsto j], Z, X K), \quad \text{and} \quad Q^j = \text{tr}(\emptyset, Z[z' \mapsto j], R[K/y, z'/z]).$$

First, we prove that  $P^j \rightarrow_{\mathcal{G}^\dagger} \{Q^j\}$ . By definition we have that

$$P^j = X_{[1 \mapsto j]}^\dagger (\text{tr}(B, Z, K))_{B \in 2^{\text{gar}(K)}},$$

and by Lemma 5.3 we have that

$$\begin{aligned} Q^j &= \text{tr}(\emptyset, Z[z' \mapsto j], R[z'/z])[\text{tr}(B, Z[z' \mapsto j], K)/y_B^\dagger \mid B \in 2^{\text{gar}(K)}] \\ &= \text{tr}(\emptyset, [z \mapsto j], R)[\text{tr}(B, Z, K)/y_B^\dagger \mid B \in 2^{\text{gar}(K)}], \end{aligned}$$

where the second equality holds because the  $z'$  does not appear in  $K$  and the variables from  $\text{dom}(Z)$  do not appear in  $R$ . Recalling that the rule for  $X_A^\dagger$  is

$$X_{[1 \mapsto j]}^\dagger (y_B^\dagger)_{B \in 2^{\text{gar}(y)}} \rightarrow \text{tr}(\emptyset, [z \mapsto j], R),$$

we immediately see that indeed  $P^j \rightarrow_{\mathcal{G}^\dagger} \{Q^j\}$ . Having this, we recall that

$$\text{tr}(\emptyset, Z, E) = \oplus \langle P^0, \bullet \langle P^1, L' \rangle \rangle \quad \text{and} \quad \text{tr}(\emptyset, Z, F) = \oplus \langle Q^0, \bullet \langle Q^1, L' \rangle \rangle \quad (1)$$

for appropriate  $L'$  (obtained by transforming  $L$ ). Recall that, by definition, a term  $M$  is  $\mathcal{G}^\dagger$ -convergent if and only if  $M \rightarrow_{\mathcal{G}^\dagger} \mathcal{N}$  for some set  $\mathcal{N}$  of  $\mathcal{G}^\dagger$ -convergent terms. Thus, the only way why  $\text{tr}(\emptyset, Z, F)$  can be  $\mathcal{G}^\dagger$ -convergent (which holds by assumption) is that either  $Q^0$  is  $\mathcal{G}^\dagger$ -convergent, or both  $Q^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent. Because of the reduction  $P^j \rightarrow_{\mathcal{G}^\dagger} \{Q^j\}$  we have that either  $P^0$  is  $\mathcal{G}^\dagger$ -convergent, or both  $P^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent, which implies that  $\text{tr}(\emptyset, Z, E)$  is  $\mathcal{G}^\dagger$ -convergent.

In Cases (2) and (3), when  $E = \bullet \langle K_1, \dots, K_k \rangle$  or  $E = \oplus \langle K_1, \dots, K_k \rangle$ , we have a reduction from  $\text{tr}(\emptyset, Z, E)$  to  $\{\text{tr}(\emptyset, Z, F) \mid F \in \mathcal{F}\}$ , because  $\text{tr}$  distributes over  $\bullet \langle \dots \rangle$  and  $\oplus \langle \dots \rangle$ . In Cases (4) and (5) (elimination of explicit substitution) we also have similar reductions.

Finally, in Case (6) we have that

$$E = E_0 \langle L/z \rangle, \quad \mathcal{F} = \{E_1 \langle L/z \rangle, \dots, E_k \langle L/z \rangle\}, \quad \text{and} \quad E_0 \rightsquigarrow_{\mathcal{G}} \{E_1, \dots, E_k\}.$$

By definition, for every  $i \in \{0, \dots, k\}$  we have that

$$\begin{aligned} \text{tr}(\emptyset, Z, E_i \langle L/z \rangle) &= \oplus \langle P_i^0, \bullet \langle P_i^1, L' \rangle \rangle, \quad \text{where} \\ P_i^0 &= \text{tr}(\emptyset, Z[z \mapsto 0], E_i), \quad P_i^1 = \text{tr}(\emptyset, Z[z \mapsto 1], E_i), \quad L' = \text{tr}(\emptyset, Z, L). \end{aligned} \quad (2)$$

Thus,  $\text{tr}(\emptyset, Z, E_i \langle L/z \rangle)$  is  $\mathcal{G}^\dagger$ -convergent if and only if either  $P_i^0$  is  $\mathcal{G}^\dagger$ -convergent, or both  $P_i^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent. By assumption this is the case for all  $i \in [k]$ , and we have to prove this for  $i = 0$ . If for every  $i \in [k]$  we have the former case (i.e.,  $P_i^0$  is  $\mathcal{G}^\dagger$ -convergent), by the

induction hypothesis (used with the function  $Z[z \mapsto 0]$ ) we have that  $P_0^0$  is  $\mathcal{G}^\dagger$ -convergent, and we are done. In the opposite case, for some  $i \in [k]$  (but for at least one of them) we have that both  $P_i^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent, and for the remaining  $i \in [k]$  we have that  $P_i^0$  is  $\mathcal{G}^\dagger$ -convergent. Using Lemma 5.4 we deduce that if  $P_i^0$  is  $\mathcal{G}^\dagger$ -convergent, then also  $P_i^1$  is  $\mathcal{G}^\dagger$ -convergent. Thus actually  $P_i^1$  is  $\mathcal{G}^\dagger$ -convergent for every  $i \in [k]$ , and additionally  $L'$  is  $\mathcal{G}^\dagger$ -convergent. By the induction hypothesis (used with the function  $Z[z \mapsto 1]$ ) we have that  $P_0^1$  is  $\mathcal{G}^\dagger$ -convergent, and we are also done.  $\blacktriangleleft$

We can now conclude with the left-to-right direction of Lemma 5.2:

► **Lemma 5.6.** *Let  $E$  be an extended term over  $(\mathcal{X}, \emptyset)$ . If  $E$  is  $\mathcal{G}$ -ext-convergent, then  $\text{tr}(\emptyset, \emptyset, E)$  is  $\mathcal{G}^\dagger$ -convergent.*

**Proof.** Induction on the fact that  $E$  is  $\mathcal{G}$ -ext-convergent. Because  $E$  is  $\mathcal{G}$ -ext-convergent,  $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$  for some set  $\mathcal{F}$  of  $\mathcal{G}$ -ext-convergent extended terms, for which we can apply the induction hypothesis. The induction hypothesis says that  $\text{tr}(\emptyset, \emptyset, F)$  is  $\mathcal{G}^\dagger$ -convergent for every  $F \in \mathcal{F}$ . In such a situation Lemma 5.5 implies that  $\text{tr}(\emptyset, \emptyset, E)$  is also  $\mathcal{G}^\dagger$ -convergent, as required.  $\blacktriangleleft$

For a proof in the opposite direction we need the following definition. We say that a term  $M$  is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps if  $M \rightarrow_{\mathcal{G}^\dagger} \{N_1, \dots, N_k\}$ , and every  $N_i$  is  $\mathcal{G}^\dagger$ -convergent in  $n_i$  steps, and  $n = 1 + n_1 + \dots + n_k$  (i.e., we count 1 for the above reduction, and we sum the numbers of steps needed to reduce all  $N_i$ ). Clearly a term  $M$  is  $\mathcal{G}^\dagger$ -convergent if and only if it is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps for some  $n \in \mathbb{N}$ . Notice that the number  $n$  is not determined by  $M$  (i.e., that the same term  $M$  can be  $\mathcal{G}^\dagger$ -convergent in  $n$  steps for multiple values of  $n$ ). We can now state the converse of Lemma 5.5:

► **Lemma 5.7.** *Let  $E$  be an extended term over  $(\mathcal{X}, \mathcal{Z})$  and let  $Z \in 2^{\mathcal{Z}}$ . If  $\text{tr}(\emptyset, Z, E)$  is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps and  $E$  is not a variable, then there exists a set  $\mathcal{F}$  of extended terms such that  $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$  and  $\text{tr}(\emptyset, Z, F)$  is  $\mathcal{G}^\dagger$ -convergent in less than  $n$  steps for every  $F \in \mathcal{F}$ .*

**Proof.** Induction on the number of explicit substitutions in  $E$ . Depending on the shape of  $E$ , we have several cases. Missing details are given in the full version of the paper (Appendix C).

One case is  $E$  consists of a nonterminal  $X$  with some arguments applied. For simplicity of presentation, we again suppose that  $X$  has two arguments:  $y$  of positive order, and  $z$  of order 0. Thus,  $E$  is of the form  $E = X K L$ . Let  $X y z \rightarrow R$  be the rule for  $X$ , and let  $z'$  be a fresh variable of type  $\circ$  not appearing in  $\mathcal{Z}$ . In such a situation, taking  $F = R[K/y, z'/z]\langle L/z' \rangle$  we have that  $E \rightsquigarrow_{\mathcal{G}} \{F\}$ . Recall the terms  $P^j$  and  $Q^j$  (for  $j \in \{0, 1\}$ ) from the proof of Lemma 5.5. In that proof we have observed that  $P^j \rightarrow_{\mathcal{G}^\dagger} \{Q^j\}$ . But clearly this is the only way how  $P^j$  can reduce, so if  $P^j$  is  $\mathcal{G}^\dagger$ -convergent in  $n_j$  steps, then necessarily  $Q^j$  is  $\mathcal{G}^\dagger$ -convergent in  $n_j - 1$  steps. By Equalities (1) we have that if  $\text{tr}(\emptyset, Z, E)$  is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps, then either  $P^0$  is  $\mathcal{G}^\dagger$ -convergent in  $n_0 = n - 1$  steps, or both  $P^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent in, respectively,  $n_1$  and  $n - n_1 - 2$  steps, for some  $n_1 \in \mathbb{N}$ . In the former case,  $Q^0$  is  $\mathcal{G}^\dagger$ -convergent in  $n_0 - 1 = n - 2$  steps, so  $\text{tr}(\emptyset, Z, F)$  is  $\mathcal{G}^\dagger$ -convergent in  $n - 1$  steps, and we are done. In the latter case,  $Q^1$  is  $\mathcal{G}^\dagger$ -convergent in  $n_1 - 1$  steps, so  $\text{tr}(\emptyset, Z, F)$  is  $\mathcal{G}^\dagger$ -convergent in  $(n_1 - 1) + (n - n_1 - 2) + 2 = n - 1$  steps, and we are done again.

Notice that we do not have a similar case for a variable with some arguments applied, because the whole  $E$  is not a variable, and because (by definition of an extended term) all free variables of  $E$  are of type  $\circ$ .

The cases of  $E = \bullet\langle K_1, \dots, K_k \rangle$  and  $E = \oplus\langle K_1, \dots, K_k \rangle$  are straightforward.

## 53:12 Higher-Order Nonemptiness Step by Step

It remains to assume that  $E$  is an explicit substitution. If  $E = z\langle L/z \rangle$ , we should take  $\mathcal{F} = \{L\}$ , and if  $E = z'\langle L/z \rangle$  for  $z' \neq z$ , we should take  $\mathcal{F} = \{z'\}$  (in these two subcases we cannot use the induction assumption, because it does not work for an extended term being a single variable). Otherwise  $E = E_0\langle L/z \rangle$ , where  $E_0$  is not a variable. Recall that  $\text{tr}(\emptyset, Z, E) = \oplus \langle P_0^0, \bullet \langle P_0^1, L' \rangle \rangle$  for  $P_0^0, P_0^1, L'$  as in the proof of Lemma 5.5. By assumption  $\text{tr}(\emptyset, Z, E)$  is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps, so either  $P_0^0$  is  $\mathcal{G}^\dagger$ -convergent in  $n' = n - 1$  steps, or both  $P_0^1$  and  $L'$  are  $\mathcal{G}^\dagger$ -convergent in, respectively,  $n'$  and  $n - n' - 2$  steps, for some  $n' \in \mathbb{N}$ . Let  $j = 0$  in the former case and  $j = 1$  in the latter case. The induction hypothesis gives us a set  $\{E_1, \dots, E_k\}$  such that  $E_0 \rightsquigarrow_{\mathcal{G}} \{E_1, \dots, E_k\}$  and  $\text{tr}(\emptyset, Z[z \mapsto j], E_i)$  is  $\mathcal{G}^\dagger$ -convergent in less than  $n'$  steps for every  $i \in [k]$ . We then take

$$\mathcal{F} = \{E_1\langle L/z \rangle, \dots, E_k\langle L/z \rangle\}.$$

Equality (2) holds now for all  $i \in \{0, \dots, k\}$ . For  $j = 0$  we use that the fact that  $\text{tr}(\emptyset, Z, E_i\langle L/z \rangle) \rightarrow_{\mathcal{G}^\dagger} \{P_i^0\}$ , which implies that  $\text{tr}(\emptyset, Z, E_i\langle L/z \rangle)$  is  $\mathcal{G}^\dagger$ -convergent in less than  $n' + 1 = n$  steps, as required. For  $j = 1$  we use that the fact that  $\text{tr}(\emptyset, Z, E_i\langle L/z \rangle) \rightarrow_{\mathcal{G}^\dagger} \{\bullet \langle P_i^1, L' \rangle\}$  and  $\bullet \langle P_i^1, L' \rangle \rightarrow_{\mathcal{G}^\dagger} \{P_i^1, L'\}$ , which implies that  $\text{tr}(\emptyset, Z, E_i\langle L/z \rangle)$  is  $\mathcal{G}^\dagger$ -convergent in less than  $n' + (n - n' - 2) + 2 = n$  steps, as required.  $\blacktriangleleft$

The next lemma finishes the proof of Lemma 5.2, and thus the proof of correctness of our transformation:

**► Lemma 5.8.** *Let  $E$  be an extended term over  $(\mathcal{X}, \emptyset)$ . If  $\text{tr}(\emptyset, \emptyset, E)$  is  $\mathcal{G}^\dagger$ -convergent then  $E$  is  $\mathcal{G}$ -ext-convergent.*

**Proof.** Induction on the (smallest) number  $n$  such that  $\text{tr}(\emptyset, \emptyset, E)$  is  $\mathcal{G}^\dagger$ -convergent in  $n$  steps. By assumption  $E$  is not a variable, because it is an extended term over  $(\mathcal{X}, \emptyset)$  (no free variables). So, by Lemma 5.5 there exists a set  $\mathcal{F}$  of extended terms such that  $E \rightsquigarrow_{\mathcal{G}} \mathcal{F}$  and  $\text{tr}(\emptyset, \emptyset, F)$  is  $\mathcal{G}^\dagger$ -convergent in less than  $n$  steps for every  $F \in \mathcal{F}$ . By the induction hypothesis every  $F \in \mathcal{F}$  is  $\mathcal{G}$ -ext-convergent, so by definition also  $E$  is  $\mathcal{G}$ -ext-convergent.  $\blacktriangleleft$

## 6 Conclusions

We have presented a new, simple algorithm checking whether a higher-order grammar generates a nonempty language. One may ask whether this algorithm can be used in practice. Of course the complexity  $n$ -EXPTIME for grammars of order  $n$  is unacceptably large (even if we take into account the fact that we are  $n$ -fold exponential only in the arity of types, not in the size of a grammar), but one has to recall that there exist tools solving the considered problem in such a complexity. The reason why these tools work is that the time spent by them on “easy” inputs is much smaller than the worst-case complexity (and many “typical inputs” are indeed easy). Unfortunately, this is not the case for our algorithm: the size of the grammar resulting from our transformation is always large, even if the original grammar generated a nonempty (or empty) language for some “easy reason”. Thus, our algorithm is mainly of a theoretical interest.

The presented transformation preserves nonemptiness, and thus can be used to solve the nonemptiness problem for higher-order grammars. However, it seems feasible that other problems concerning higher-order grammars (higher-order recursion schemes), like model-checking against parity automata or the simultaneous unboundedness problem [7], can be solved using similar transformations. Developing such transformations is a possible direction for further work.

## References

- 1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Log. Methods Comput. Sci.*, 3(3), 2007. doi:10.2168/LMCS-3(3:1)2007.
- 2 Kazuyuki Asada and Naoki Kobayashi. Size-preserving translations from order-(n+1) word grammars to order-n tree grammars. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPICs*, pages 22:1–22:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.FSCD.2020.22.
- 3 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.
- 4 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 129–148. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.129.
- 5 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1\_9.
- 6 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.
- 7 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. *CoRR*, abs/1605.00371, 2016. arXiv:1605.00371.
- 8 Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.
- 9 Matthew Hague, Roland Meyer, Sebastian Muskalla, and Martin Zimmermann. Parity to safety in polynomial time for pushdown and collapsible pushdown systems. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPICs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.57.
- 10 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.
- 11 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6\_15.
- 12 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 416–428. ACM, 2009. doi:10.1145/1480881.1480933.

- 13 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.
- 14 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.
- 15 Naoki Kobayashi and C.-H. Luke Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:9)2011.
- 16 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi:10.1016/j.ic.2014.12.015.
- 17 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.
- 18 Pawel Parys. Recursion schemes and the WMSO+U logic. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPICs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.53.
- 19 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.
- 20 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.
- 21 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPICs*, pages 229–243. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.CSL.2015.229.
- 22 Taku Terao and Naoki Kobayashi. A ZDD-based efficient higher-order model checking algorithm. In Jacques Garrigue, editor, *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 354–371. Springer, 2014. doi:10.1007/978-3-319-12736-1\_19.

# The Degree of a Finite Set of Words

**Dominique Perrin**

Université Gustave Eiffel, LIGM, Marne-la-Vallée, France  
dominique.perrin@esiee.fr

**Andrew Ryzhikov** 

Université Gustave Eiffel, LIGM, Marne-la-Vallée, France  
ryzhikov.andrew@gmail.com

---

## Abstract

We generalize the notions of the degree and composition from uniquely decipherable codes to arbitrary finite sets of words. We prove that if  $X = Y \circ Z$  is a composition of finite sets of words with  $Y$  complete, then  $d(X) \leq d(Y) \cdot d(Z)$ , where  $d(T)$  is the degree of  $T$ . We also show that a finite set is synchronizing if and only if its degree equals one.

This is done by considering, for an arbitrary finite set  $X$  of words, the transition monoid of an automaton recognizing  $X^*$  with multiplicities. We prove a number of results for such monoids, which generalize corresponding results for unambiguous monoids of relations.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Formal languages and automata theory

**Keywords and phrases** synchronizing set, degree of a set, group of a set, monoid of relations

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.54

**Acknowledgements** We thank Jean-Eric Pin and Jacques Sakarovitch for references concerning the composition of automata and transducers.

## 1 Introduction

Let  $X$  be a set of finite words. The set  $X^*$  of all concatenations of words in  $X$  (often called the Kleene star of  $X$ ) plays an important role in formal languages theory and its applications. The set  $X$  often represents a dictionary or a code transmitted over a channel, so the case where  $X$  is finite is especially important. In general, a word in  $X^*$  can have several different factorizations over  $X$ , and it is useful to understand the relations between them. A word  $w$  is called *synchronizing* for  $X$  if for any words  $u, v$  such that  $uwv \in X^*$  we have  $uw, vw \in X^*$ . In particular, we get that any word in  $X^*$  containing  $w$  as a factor, that is, any word of the form  $uwv$ , has a factorization where  $uw$  and  $vw$  are both in  $X^*$ , and thus can be factorized separately. A set which admits a synchronizing word is also called *synchronizing*. A set  $X$  is called *complete* if every word over the same alphabet occurs as a factor of a word in  $X^*$ .

Synchronizing words are studied a lot for uniquely decipherable codes (see e.g., Chapter 10 of [3]). A set  $X$  of words is called a *uniquely decipherable code* (often also called a *variable length code*) if every word has at most one factorization over  $X$ . Such codes play a crucial role in the theory of data compression and transmission [3].

Provided a set  $Z$  of words such that  $X \subset Z^*$ , one can rewrite  $X$  using  $Z$  as the alphabet, thus resulting in a new set  $Y$ . The representation  $X = Y \circ Z$  is then called a decomposition of  $X$ , and the converse process of obtaining  $X$  is called composition. Decomposition of a set allows to represent it by using simpler sets as building blocks, while preserving many properties of the initial one. Conversely, compositions of codes allow to construct more complicated codes by using simple ones, so they are interesting on their own. In particular, the composition of two uniquely decipherable codes is again a uniquely decipherable code [3]. For any injective morphism  $\alpha : A^* \rightarrow B^*$ ,  $\alpha(A)$  is a code, and each code can be obtained as the image of  $A$  for some  $A$  and  $\alpha$  [3]. Compositions of codes are then nothing more than



© Dominique Perrin and Andrew Ryzhikov;  
licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 54; pp. 54:1–54:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

compositions of injective morphisms between free monoids. The notion of composition of two arbitrary finite sets of words is also natural as it corresponds to the composition of arbitrary morphisms.

**Our contributions.** In this paper, we transfer the notions of composition, degree and group from uniquely decipherable codes to arbitrary finite sets of words. This extends the presentation of [3] made for uniquely decipherable codes.

Provided a finite set  $X$  of words, we associate a special automaton  $\mathcal{A}$  (called the flower automaton) recognizing  $X^*$  with multiplicities. Let  $S$  be the set of fixed points of an idempotent  $e$  of minimum rank in the transition relation of  $\mathcal{A}$ , and  $\Gamma$  be the set of strongly connected components of  $S$ . We consider a permutation group  $G_e$  acting on  $\Gamma$ . We show that all such groups are equivalent for idempotents of minimum rank (Theorem 20). Moreover, we show that for a given  $X$  all these groups are equivalent for any trim automaton recognizing  $X^*$  with multiplicities (Proposition 21). Thus this group is an invariant of a set. We introduce the degree  $d(X)$  of  $X$ , which is the minimum rank of elements in the transition monoid of  $\mathcal{A}$ . We then show that synchronizing sets are exactly sets of degree one (Proposition 22). As our main contribution, we use the obtained results to show that for a composition  $X = Y \circ Z$  of two finite sets  $Y, Z$  such that  $Y$  is complete we have  $d(X) \leq d(Y) \cdot d(Z)$  (Theorem 24).

For a finite set  $X$ , all these results were previously known only for the special case of  $X$  being a uniquely decipherable code with the equality  $d(X) = d(Y)d(Z)$  instead of an inequality [3]. Our generalization to the case of an arbitrary finite set requires more complicated proofs. In particular, for uniquely decipherable codes it is enough to consider a trim unambiguous automaton recognizing  $X^*$  (which is a cornerstone of the theory), while in our case we need a trim automaton recognizing  $X^*$  with multiplicities. Intuitively, such automata count the number of factorizations over  $X$ , and thus they are unambiguous when  $X$  is a uniquely decipherable code. The technical difficulties then begin with the replacement of unambiguous monoids of relations by arbitrary monoids of relations. Indeed, the multiplication of matrices there is different from the result over the Boolean semiring. In particular, the representation of maximal subgroups by permutations is still possible but more complicated.

**Motivation and related results.** Larger classes of codes are considered both in theory and in practice. Particular examples include multiset and set decipherable codes. A set  $X$  of words is called a *multiset* [10] (respectively, *set* [13]) *decipherable code* if every factorization of a word into codewords provides the same multiset (respectively, set) of codewords. Such codes are used if one needs to transmit only the frequencies (or the fact of occurrences) of elements, but the order of these elements does not matter. Lempel [13] reports online compilations of inventories, construction of histograms, or updating of relative frequencies as particular examples. An important property of multiset decipherable codes is that there exist examples of such codes with Kraft-McMillan sum more than one, which shows that such codes can be more efficient than uniquely decipherable codes [18]. An even wider class is that of numerically decipherable codes, which are sets with the property that every factorization of a word over such set has the same number of codewords [21]. A similar setting of multivalued encodings allows to have several different codewords for the same symbol [4]. In view of that, the transit of results from uniquely decipherable codes to arbitrary sets is interesting.

Another motivation for studying factorizations of words in  $X^*$  for an arbitrary finite set  $X$  is the area of static dictionary compression, where one looks for some specific factorization of a text over some finite dictionary [1]. The dictionary does not have to be a uniquely

decipherable code, thus a text can have several different factorizations. In this case, it is useful to know the relation between different factorizations. The parallel version of this problem is also considered [15]. In [6] a fast algorithm for checking if a given word  $w$  belongs to  $X^*$  is suggested. If the answer is positive, it also provides a factorization of  $w$  over  $X$ .

Only few results are known about decompositions and synchronization of arbitrary sets of words. The defect theorem states that every finite set of words which is not a uniquely decipherable code can be decomposed over a set of smaller size [2]. A survey of different generalizations of this theorem is presented in [11]. Synchronization in arbitrary monoids was studied in [5] and [7]. Other properties of factorizations are studied in [17, 20].

**Organization of the paper.** To transfer the results from uniquely decipherable code to arbitrary finite sets of words, we first set a correspondence with an adequate class of automata, namely automata recognizing with multiplicities (Sections 2 and 3). Then we introduce the notion of a composition for arbitrary finite sets of words (Section 4). We extend the theory of unambiguous monoids of relations by the theory of arbitrary monoids of relations (Section 5), and generalize the notion of the group  $G(X)$  and the degree  $d(X)$  of a finite set  $X$  of words (Section 6). In this way, as for codes, a set is synchronizing if and only if it is of degree 1 (Section 7). As the main result, we prove that if  $X = Y \circ Z$  with  $Y$  complete, then  $d(X) \leq d(Y) \cdot d(Z)$  (Section 8). In Section 9 we show that if we require  $Y$  to be complete, we do not get any new decompositions of a uniquely decipherable code other than into two uniquely decipherable codes.

## 2 Automata

We denote by  $A^*$  the free monoid on a finite alphabet  $A$ , by  $1$  the empty word, and by  $A^+$  the set  $A^* \setminus \{1\}$ . For notions not defined in this section see [3].

Let  $\mathcal{A} = (Q, i, t)$  be an automaton on the alphabet  $A$  with  $Q$  as set of states,  $i$  as initial state and  $t$  as terminal state (we will not need to have several initial or terminal states). We do not specify in the notation the set of edges, which are triples  $(p, a, q)$  with two states  $p, q \in Q$  and a label  $a \in A$  denoted  $p \xrightarrow{a} q$ . We form paths as usual by concatenating consecutive edges. An automaton is called *trim* if there exists a path from  $i$  to every state, and from every state to  $t$ .

The *language recognized* by  $\mathcal{A}$ , denoted  $L(\mathcal{A})$ , is the set of words in  $A^*$  which are labels of paths from  $i$  to  $t$ . There can be several paths from  $i$  to  $t$  for a given label, and this motivates the introduction of multiplicities.

For a semiring  $K$ , a  $K$ -subset of  $A^*$  is a map from  $A^*$  into  $K$ . The value of a  $K$ -subset  $X$  at  $w$  is called its *multiplicity* and denoted  $(X, w)$ . We denote by  $K\langle\langle A \rangle\rangle$  the semiring of  $K$ -subsets of  $A^*$  and by  $K\langle A \rangle$  the set of corresponding polynomials, that is the  $K$ -subsets with a finite number of words with nonzero multiplicity (on these notions, see [8]).

If  $X, Y$  are  $K$ -subsets, then  $X + Y$  and  $XY$  are the  $K$ -subsets defined by

$$(X + Y, w) = (X, w) + (Y, w), \quad (XY, w) = \sum_{w=uv} (X, u)(Y, v).$$

Moreover, if  $X$  does not have a constant term, that is, if  $(X, 1) = 0$ , then  $X^*$  is the  $K$ -subset

$$X^* = 1 + X + X^2 + \dots$$

Since  $X$  has no constant term, for every word  $w$ , the number of nonzero terms  $(X^n, w)$  in the sum above is finite and thus  $X^*$  is well-defined.

54:4 The Degree of a Finite Set of Words

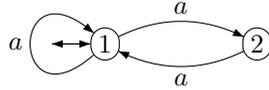
For a set  $X \subset A^*$ , we denote by  $\underline{X}$  the characteristic series of  $X$ , considered as an  $\mathbb{N}$ -subset. It is easy to verify that for  $X \subset A^+$ , the multiplicity of  $w \in A^*$  in  $\underline{X}$  is the number of factorizations of  $w$  in words of  $X$ .

For an automaton  $\mathcal{A} = (Q, i, t)$  on the alphabet  $A$ , we denote by  $|\mathcal{A}|$  its behaviour, which is an element of  $\mathbb{N}\langle\langle A \rangle\rangle$ . It is the  $\mathbb{N}$ -subset of  $A^*$  such that the multiplicity of  $w \in A^*$  in  $|\mathcal{A}|$  is the number of paths from  $i$  to  $t$  labeled  $w$  in  $\mathcal{A}$ .

We denote by  $\mu_{\mathcal{A}}$  the morphism from  $A^*$  into the monoid of  $Q \times Q$ -matrices with integer coefficients defined for  $\mu_{\mathcal{A}}(w)_{p,q}$  as the number of paths from  $p$  to  $q$  labeled by  $w$ . Thus, the multiplicity of  $w$  in  $|\mathcal{A}|$  is  $(|\mathcal{A}|, w) = \mu_{\mathcal{A}}(w)_{i,t}$ .

Given a set  $X \subset A^+$ , we say that the automaton  $\mathcal{A}$  recognizes  $X^*$  with multiplicities if the behaviour of  $\mathcal{A}$  is the multiset assigning to  $x$  its number of distinct factorizations in  $X$ . Formally,  $\mathcal{A}$  recognizes  $X^*$  with multiplicities if  $|\mathcal{A}| = \underline{X}^*$ .

► **Example 1.** Let  $X = \{a, a^2\}$ . The number of factorizations of  $a^n$  in words of  $X$  is the Fibonacci number  $F_{n+1}$  defined by  $F_0 = 0, F_1 = 1$  and  $F_{n+1} = F_n + F_{n-1}$  for  $n \geq 1$ . The automaton  $\mathcal{A}$  represented in Figure 1 recognizes  $X^*$  with multiplicities, that is  $|\mathcal{A}| = (a + a^2)^*$ .



■ **Figure 1** An automaton recognizing  $X^*$  with multiplicities.

We have indeed for every  $n \geq 1$ ,

$$\mu_{\mathcal{A}}(a^n) = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

For an automaton  $\mathcal{A} = (Q, i, t)$  on the alphabet  $A$ , we denote by  $\varphi_{\mathcal{A}}$  the morphism from  $A^*$  onto the monoid of transitions of  $\mathcal{A}$ . Thus, for  $w \in A^*$ ,  $\varphi_{\mathcal{A}}(w)$  is the Boolean  $Q \times Q$ -matrix defined by

$$\varphi_{\mathcal{A}}(w)_{p,q} = \begin{cases} 1 & \text{if } p \xrightarrow{w} q, \\ 0 & \text{otherwise} \end{cases}$$

Let  $X \subset A^+$  be a finite set of words on the alphabet  $A$ . The *flower automaton* of  $X$  is the following automaton. Its set of states is the subset  $Q$  of  $A^* \times A^*$  defined as

$$Q = \{(u, v) \in A^+ \times A^+ \mid uv \in X\} \cup (1, 1).$$

We often denote  $\omega = (1, 1)$ . There are four type of edges labeled by  $a \in A$

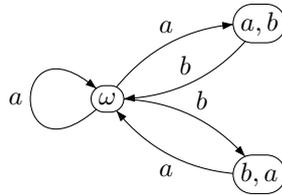
$$\begin{aligned} (u, av) &\xrightarrow{a} (ua, v) && \text{for } uav \in X, u, v \neq 1 \\ \omega &\xrightarrow{a} (a, v) && \text{for } av \in X, v \neq 1 \\ (u, a) &\xrightarrow{a} \omega && \text{for } ua \in X, u \neq 1 \\ \omega &\xrightarrow{a} \omega && \text{for } a \in X. \end{aligned}$$

The state  $\omega$  is both initial and terminal.

The proof of the following result is straightforward. It generalizes the fact that the flower automaton of a code recognizes  $X^*$  and is unambiguous (see Theorem 4.2.2 in [3]).

► **Proposition 2.** For any finite set  $X \subset A^+$ , the flower automaton of  $X$  recognizes  $X^*$  with multiplicities.

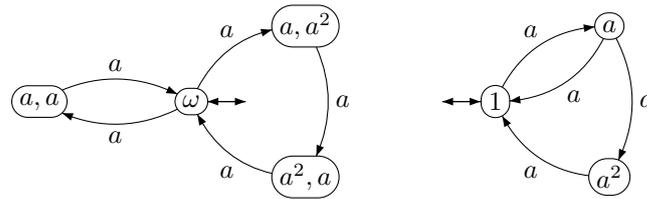
► **Example 3.** Let  $X = \{a, ab, ba\}$ . The flower automaton of  $X^*$  is represented in Figure 2. As an example, there are two paths from  $\omega$  to  $\omega$  labeled  $aba$ , corresponding to the two factorizations  $(a)(ba) = (ab)(a)$ .



■ **Figure 2** The flower automaton of  $X$  (Example 3).

A more compact version of the flower automaton is the *prefix automaton*  $\mathcal{A} = (P, 1, 1)$  of a finite set  $X \subset A^+$ . Its set of states is the set  $P$  of proper prefixes of  $X$  and its edges are the  $p \xrightarrow{a} pa$  for every  $p \in P$  and  $a \in A$  such that  $pa \in P$  and the  $p \xrightarrow{a} 1$  such that  $pa \in X$ . It also recognizes  $X^*$  with multiplicities.

► **Example 4.** Let  $X = \{a^2, a^3\}$ . The flower automaton of  $X$  is shown in Figure 3 on the left and its prefix automaton on the right.



■ **Figure 3** The flower automaton and the prefix automaton of  $X$  (Example 4).

A *reduction* from an automaton  $\mathcal{A} = (P, i, t)$  onto an automaton  $\mathcal{B} = (Q, j, u)$  is a surjective map  $\rho : P \rightarrow Q$  such that  $\rho(i) = j$ ,  $\rho(t) = u$  and such that for every  $q, q' \in Q$  and  $w \in A^*$ , there is a path  $q \xrightarrow{w} q'$  in  $\mathcal{B}$  if and only if there is a path  $p \xrightarrow{w} p'$  in  $\mathcal{A}$  for some  $p, p' \in P$  with  $\rho(p) = q$  and  $\rho(p') = q'$ .

The reduction is *sharp* if  $\rho^{-1}(j) = \{i\}$  and  $\rho^{-1}(u) = \{t\}$ .

► **Proposition 5.** Let  $\rho$  be a reduction from  $\mathcal{A} = (P, i, t)$  onto  $\mathcal{B} = (Q, j, u)$ . Then  $L(\mathcal{A}) \subset L(\mathcal{B})$ , with equality if  $\rho$  is sharp.

The term reduction is the one used in [3] and it is not standard but captures the general idea of a covering. The term conformal morphism is the one used in [19]. The following statement replaces [3, Proposition 4.2.5].

► **Proposition 6.** Let  $X \subset A^+$  be a finite set which is the minimal generating set of  $X^*$ . For each trim automaton  $\mathcal{B} = (Q, i, i)$  recognizing  $X^*$  with multiplicities, there is a sharp reduction from the flower automaton of  $X$  onto  $\mathcal{B}$ .

**Proof.** Let  $\mathcal{A} = (P, \omega, \omega)$  be the flower automaton of  $X$ . We define a map  $\rho : P \rightarrow Q$  as follows. We set first  $\rho(\omega) = i$ . Next, if  $(u, v) \in P$  with  $(u, v) \neq \omega$ , then  $uv \in X$ . Since  $X$  is

the minimal generating set of  $X^*$ , there is only one factorization of  $uv$  into elements of  $X$ . Since  $\mathcal{B}$  recognizes  $X$  with multiplicities, there is only one path  $i \xrightarrow{u} q \xrightarrow{v} i$  in  $\mathcal{B}$ . We define  $\rho((u, v)) = q$ .

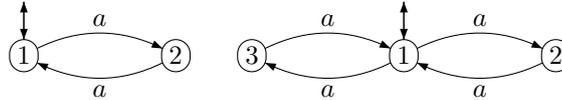
It is straightforward to verify that  $\rho$  is a reduction. Assume first that  $q \xrightarrow{w} q'$  in  $\mathcal{B}$ . Let  $i \xrightarrow{u} q$  and  $q' \xrightarrow{v'} i$  be simple paths, that is not passing by  $i$  except at the origin or the end. Then  $i \xrightarrow{u v'} i$  and thus  $u v' = x_1 x_2 \cdots x_n$  with  $x_i \in X$ ,  $u$  a proper prefix of  $x_1 = uv$  and  $v'$  a proper suffix of  $x_n = u'v'$ . Thus  $\rho((u, v)) = q$  and  $\rho((u', v')) = q'$ . Since  $w = v x_2 \cdots x_{n-1} u'$ , we have in  $\mathcal{A}$  a path  $(u, v) \xrightarrow{w} (u', v')$ . Conversely, consider a path  $(u, v) \xrightarrow{w} (u', v')$  in  $\mathcal{A}$ . If the path does not pass by  $\omega$ , then  $u' = uw$ ,  $v = wv'$  and we have a path  $q \xrightarrow{w} q'$  in  $\mathcal{B}$  with  $\rho((u, v)) = q$  and  $\rho((u', v')) = q'$ . Otherwise, the path decomposes in  $(u, v) \xrightarrow{v} \omega \xrightarrow{x} \omega \xrightarrow{v'} (u', v')$  with  $x \in X^*$ . Since  $\mathcal{B}$  recognizes  $X^*$ , we have a path  $i \xrightarrow{x} i$  in  $\mathcal{B}$  and thus also a path  $q \xrightarrow{w} q'$  with  $q = \rho((u, v))$  and  $q' = \rho((u', v'))$ . ◀

The statement above is false if  $X$  is not the minimal generating set of  $X^*$ , as shown by the following example.

► **Example 7.** Let  $X = \{a, a^2\}$ . There is no sharp reduction from the automaton of Figure 1 onto the one-state automaton recognizing  $X^* = \{a\}^*$ .

The statement is also false if the automaton  $\mathcal{B}$  recognizes  $X^*$ , but does not recognize  $X^*$  with multiplicities, as shown by the following example.

► **Example 8.** Let  $X = \{a^2\}$ . The flower automaton of  $X$  is represented in Figure 4 on the left. There is no reduction onto the automaton represented on the right which also recognizes  $X^*$  (but not with multiplicities).



■ **Figure 4** Two automata recognizing  $X^*$ .

### 3 Transducers

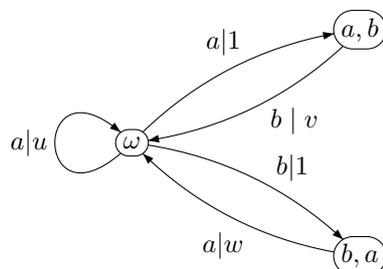
A literal *transducer*  $\mathcal{T} = (Q, i, t)$  on a set of states  $Q$  with an *input alphabet*  $A$  and an *output alphabet*  $B$  is defined by a set of edges  $E$  which are of the form  $p \xrightarrow{(a,v)} q$  with  $p, q \in Q$ ,  $a \in A$  and  $v \in B \cup \{1\}$ . The *input automaton* associated with a transducer is the automaton with the same set of states and edges but with the output labels removed.

The relation *realized* by the transducer  $\mathcal{T}$  is the set of pairs  $(u, v) \in A^* \times B^*$  such that there is a path from  $i$  to  $t$  labeled  $(u, v)$ . We denote by  $\varphi_{\mathcal{T}}$  the morphism from  $A^*$  to the monoid of  $Q \times Q$ -matrices with elements in  $\mathbb{N}\langle B \rangle$  defined for  $u \in A^*$  and  $p, q \in Q$  by  $\varphi_{\mathcal{T}}(u)_{p,q} = \sum_{p \xrightarrow{u|v} q} v$ .

Let  $X \subset A^+$  be a finite set. Let  $\beta : B^* \rightarrow A^*$  be a *coding morphism* for  $X$ , that is, a morphism whose restriction to  $B$  is a bijection onto  $X$ . The *decoding relation* for  $X$  is the relation  $\gamma = \{(u, v) \in A^* \times B^* \mid u = \beta(v)\}$ . A *decoder* for  $X$  is a literal transducer which realizes the decoding relation. The *flower transducer* associated to  $\beta$  is the literal transducer built on the flower automaton of  $X$  by adding an output label 1 to each edge  $\omega \xrightarrow{a} (a, v)$  or  $(u, av) \xrightarrow{a} (ua, v)$  and an output label  $b$  to each edge  $\omega \xrightarrow{a} \omega$  such that  $a \in X$  with  $\beta(b) = a$  or  $(u, a) \xrightarrow{a} \omega$  such that  $ua = x \in X$  with  $\beta(b) = x$ .

► **Proposition 9.** For every finite set  $X \subset A^+$  with a coding morphism  $\beta$ , the flower transducer associated to  $\beta$  is a decoder for  $X$ .

► **Example 10.** Let  $X = \{a, ab, ba\}$  and let  $\beta : u \rightarrow a, v \rightarrow ab, w \rightarrow ba$ . The flower transducer associated to  $\beta$  is represented in Figure 5. One has



■ **Figure 5** The flower transducer associated to  $\beta$ .

$$\varphi_{\mathcal{T}}(a) = \begin{bmatrix} u & 1 & 0 \\ 0 & 0 & 0 \\ w & 0 & 0 \end{bmatrix} \text{ and } \varphi_{\mathcal{T}}(b) = \begin{bmatrix} 0 & 0 & 1 \\ v & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The *prefix transducer*  $\mathcal{T} = (P, 1, 1)$  is the same modification of the prefix automaton. Its states are the proper prefixes of the elements of  $X$ . There is an edge  $p \xrightarrow{a|1} pa$  for every prefix  $p$  and every letter  $a$  such that  $pa \in P$ , and an edge  $p \xrightarrow{a|b} 1$  for every prefix  $p$  and letter  $a$  such that  $pa = \beta(b) \in X$ . Thus the input automaton of the prefix transducer of  $X$  is the prefix automaton of  $X$ .

Let  $\mathcal{B} = (Q, j, j)$  be an automaton on the alphabet  $B$  and let  $\mathcal{T} = (P, i, i)$  be a literal transducer with the input alphabet  $A$  and the output alphabet  $B$ . We build an automaton  $\mathcal{A} = \mathcal{B} \circ \mathcal{T}$  on the alphabet  $A$  as follows. Its set of states is  $Q \times P$  and for every  $a \in A$ , the matrix  $\varphi_{\mathcal{A}}(a)$  is obtained by replacing in  $\varphi_{\mathcal{T}}(a)$  the word  $w = \varphi_{\mathcal{T}}(a)_{p,q}$  by the matrix  $\varphi_{\mathcal{B}}(w)$ . The initial and terminal state is  $(j, i)$ . The automaton  $\mathcal{A}$  is also called the *wreath product* of  $\mathcal{B}$  and  $\mathcal{T}$  (see [9]). The word 1 is replaced by the identity matrix, and 0 is replaced by the zero matrix of appropriate size. An example of  $\mathcal{A} = \mathcal{B} \circ \mathcal{T}$  is provided in Example 13.

## 4 Composition

Let  $Y \subset B^+$  and  $Z \subset A^+$  be finite sets of words such that there exists a bijection  $\beta : B \rightarrow Z$ . Two such sets are called *composable*. Then  $X = \beta(Y)$  is called the *composition* of  $Y$  and  $Z$  through  $\beta$ , where  $\beta(Y) = \{\beta(y) \mid y \in Y\}$  with  $\beta$  naturally extended to the mapping  $B^* \rightarrow Z^*$ . We denote  $X = Y \circ_{\beta} Z$ . We also denote  $X = Y \circ Z$  when  $\beta$  is clear. We say that  $X = Y \circ Z$  is a *decomposition* of  $X$ .

► **Example 11.** Let  $Y = \{u, uw, vu\}$  and  $Z = \{a, ab, ba\}$  with  $\beta : u \rightarrow a, v \rightarrow ab, w \rightarrow ba$ . Then  $X = Y \circ_{\beta} Z = \{a, aba\}$ .

A decomposition  $X = Y \circ_{\beta} Z$  of a finite set  $X$  is *trim* if every letter of  $B$  appears in a word of  $Y$  and every word in  $X$  is obtained in a unique way from words in  $Y$ , that is, if the restriction of  $\beta$  to  $Y$  is injective. For any decomposition  $X = Y \circ Z$ , there are  $Y' \subset Y$  and  $Z' \subset Z$  such that  $X = Y' \circ Z'$  is trim. Indeed, if  $x \in X$  has two decompositions in words of  $Z$  as  $x = z_1 z_2 \cdots z_n = z'_1 z'_2 \cdots z'_n$ , we may remove  $\beta^{-1}(z'_1 z'_2 \cdots z'_n)$  from  $Y$  without

changing  $X$ . A finite number of these removals gives a trim decomposition. The set  $Z'$  is obtained by removing all words in  $Z$  which correspond to the letters no longer occurring in words in  $Y'$  (we also remove such letters from  $B$ ). The decomposition in Example 11 is not trim, since  $aba = \beta(uw) = \beta(vu)$ , but it can be made trim by taking  $X = Y' \circ Z'$  with  $Y' = \{u, uw\}$  and  $Z' = \{a, ba\}$ . In this case,  $Y' \subset \{u, w\}^+$ .

A set  $X \subset A^*$  is *complete* if any word in  $A^*$  is a factor of a word in  $X^*$ .

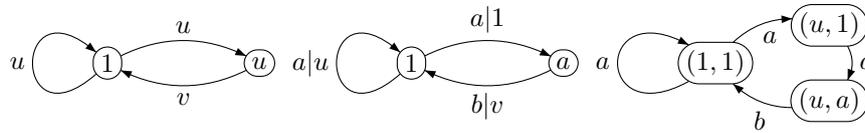
► **Proposition 12.** *Let  $Y \subset B^+$  and  $Z \subset A^+$  be two composable finite sets and let  $X = Y \circ_{\beta} Z$  be a trim decomposition. Let  $\mathcal{B} = (Q, 1, 1)$  be the prefix automaton of  $Y$  and let  $\mathcal{T} = (P, 1, 1)$  be the prefix transducer of  $Z$ . The automaton  $\mathcal{A} = \mathcal{B} \circ \mathcal{T}$  recognizes  $X^*$  with multiplicities.*

*If  $Y$  is complete, there is a reduction  $\rho$  from  $\mathcal{A}$  onto the prefix automaton of  $Z$ . Moreover, the automaton  $\mathcal{B}$  can be identified through  $\beta$  with the restriction of  $\mathcal{A}$  to  $\rho^{-1}(1)$ .*

**Proof.** The simple paths in  $\mathcal{A}$  have the form  $(1, 1) \xrightarrow{z_1} (b_1, 1) \xrightarrow{z_2} (b_1 b_2, 1) \cdots \xrightarrow{z_n} (1, 1)$  for  $x = z_1 \cdots z_n = \beta(b_1 \cdots b_n)$  in  $X$  and  $z_i \in Z$ . Since the decomposition is trim, there is exactly one such path for every  $x \in X$  and thus  $\mathcal{A}$  recognizes  $X^*$  with multiplicities.

Let us show that, if  $Y$  is complete, the map  $\rho : (q, p) \rightarrow p$  is a reduction from  $\mathcal{A}$  onto the prefix automaton of  $Z$ . We have to show that one has  $p \xrightarrow{w} p'$  in the prefix automaton  $\mathcal{C}$  of  $Z$  if and only if there exist  $q, q' \in Q$  such that  $(q, p) \xrightarrow{w} (q', p')$ . Assume that  $p \xrightarrow{w} p'$  in  $\mathcal{C}$ . Then we have  $p \xrightarrow{w|u} p'$  in the prefix transducer  $\mathcal{T}$  for some  $u \in B^*$ . Since  $Y$  is complete, there are some  $q, q' \in Q$  such that  $q \xrightarrow{u} q'$  in  $\mathcal{B}$ . Then  $(q, p) \xrightarrow{w} (q', p')$  in  $\mathcal{A}$ . The converse is obvious.

Finally, the edges of the restriction of  $\mathcal{A}$  to  $\rho^{-1}(1)$  are the simple paths  $(q, 1) \xrightarrow{z} (q', 1)$  for  $z = \beta(b) \in Z$  and  $q \xrightarrow{b} q'$  an edge of  $\mathcal{B}$ . This proves the last statement. ◀



■ **Figure 6** The prefix automaton of  $Y$ , the prefix transducer  $\mathcal{T}$  of  $Z$  and the trim part of  $\mathcal{A}$ .

► **Example 13.** Let  $Y = \{u, uv\}$  and  $Z = \{a, ab\}$  with  $\beta : u \rightarrow a, v \rightarrow ab$ . We have, in view of Figure 6,

$$\varphi_{\mathcal{A}}(a) = \begin{bmatrix} \varphi_{\mathcal{B}}(u) & I \\ 0 & 0 \end{bmatrix} \text{ and } \varphi_{\mathcal{A}}(b) = \begin{bmatrix} 0 & 0 \\ \varphi_{\mathcal{B}}(v) & 0 \end{bmatrix}.$$

## 5 Monoids of relations

We consider monoids of binary relations and prove some results on idempotents and groups in such monoids. Few authors have considered monoids of binary relations. In [16], the Green's relations in the monoid  $\mathcal{B}_Q$  of all binary relations on a set  $Q$  are considered. It is shown in [14] that any finite group appears as a maximal subgroup of  $\mathcal{B}_Q$  (in contrast with the monoid of all partial maps in which all maximal subgroups are symmetric groups).

We write indifferently relations on a set  $Q$  as subsets of  $Q \times Q$ , as boolean  $Q \times Q$ -matrices or as directed graphs on a set  $Q$  of vertices.

The *rank* of a relation  $m$  on  $Q$  is the minimal cardinality of a set  $R$  such that  $m = uv$  with  $u$  a  $Q \times R$  relation and  $v$  an  $R \times Q$  relation. Equivalently, the rank of  $m$  is the minimal number of row (resp. column) vectors (which are possibly not rows or columns of  $m$ ) which generate over  $\{0, 1\}$  the set of rows (resp. columns) of  $m$ .

For example, the full relation  $m = Q \times Q$  has rank 1. In terms of matrices

$$m = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} [1 \quad 1 \quad \cdots \quad 1]$$

More generally, the rank of an equivalence relation is equal to the number of its classes.

A *fixed point* of a relation  $m$  on  $Q$  is an element  $q \in Q$  such that  $q \xrightarrow{m} q$ . The following result appears in [20] (see also [12]).

► **Proposition 14.** *Let  $e$  be an idempotent relation on a finite set  $Q$ , let  $S$  be the set of fixed points of  $e$  and let  $\Gamma$  be the set of strongly connected components of the restriction of  $e$  to  $S$ .*

1. *For all  $p, q \in Q$  we have  $p \xrightarrow{e} q$  if and only if there exists an  $s \in S$  such that  $p \xrightarrow{e} s$  and  $s \xrightarrow{e} q$ .*
2. *We have*

$$e = \ell r \tag{1}$$

where  $\ell = \{(p, \sigma) \in Q \times \Gamma \mid p \xrightarrow{e} s \text{ for some } s \in \sigma\}$  and  $r = \{(\sigma, q) \in \Gamma \times Q \mid s \xrightarrow{e} q \text{ for some } s \in \sigma\}$ .

**Proof.** 1. Choose  $n > \text{Card}(Q)$ . Since  $p \xrightarrow{e^n} q$ , there is some  $s \in Q$  such that  $p \xrightarrow{e^i} s \xrightarrow{e^j} s \xrightarrow{e^k} q$  with  $i + j + k = n$ . Then  $p \xrightarrow{e} s \xrightarrow{e} s \xrightarrow{e} q$  and the statement is proved. The other direction is obvious.

2. If  $p \xrightarrow{e} q$ , let  $s \in S$  be such that  $p \xrightarrow{e} s \xrightarrow{e} q$  and let  $\sigma$  be the strongly connected component of  $s$ . Then  $p \xrightarrow{\ell} \sigma \xrightarrow{r} q$ . Thus  $e \leq \ell r$ , which means that each element of  $e$  is not larger than the corresponding element of  $\ell r$  when these relations are considered as binary matrices. Conversely, if  $p \xrightarrow{\ell} \sigma \xrightarrow{r} q$  there are  $s, s' \in \sigma$  such that  $p \xrightarrow{e} s$  and  $s' \xrightarrow{e} q$ . Since  $s, s'$  are in the same strongly connected component, we have  $s \xrightarrow{e} s'$  and we obtain  $p \xrightarrow{e} s \xrightarrow{e} s' \xrightarrow{e} q$ , whence  $p \xrightarrow{e} q$ . ◀

The decomposition of  $e = \ell r$  given by Equation (1) is called the *column-row decomposition* of  $e$ . Note that Proposition 14 is false without the finiteness hypothesis on  $Q$ . Indeed, the relation  $e = \{(x, y) \in \mathbb{R}^2 \mid x < y\}$  is idempotent, but has no fixed points.

► **Example 15.** The matrix

$$m = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

is an idempotent of rank 1.

For an element  $m$  of a monoid  $M$ , we denote by  $H(m)$  the  $\mathcal{H}$ -class of  $m$ , where  $\mathcal{H}$  is the Green relation  $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$  (see [3] for the definitions). It is a group if and only if it contains an idempotent  $e$  (see [3]). In this case, every  $m \in H(e)$  has a unique inverse  $m^{-1}$  in the group  $H(e)$ .

The following result is the transposition of Proposition 9.1.7 in [3] to arbitrary monoids of relations. However, the result is restricted to a statement on the group  $H(e)$  instead of the monoid  $eMe$ .

## 54:10 The Degree of a Finite Set of Words

► **Proposition 16.** *Let  $M$  be a monoid of relations on a finite set  $Q$ , let  $e \in M$  be idempotent and let  $\Gamma$  be the set of strongly connected components of the fixed points of  $e$ . For  $m \in H(e)$ , let  $\gamma_e(m)$  be the relation on  $\Gamma$  defined by*

$$\gamma_e(m) = \{(\rho, \sigma) \in \Gamma \times \Gamma \mid r \xrightarrow{m} s \xrightarrow{m^{-1}} r \text{ for some } r \in \rho \text{ and } s \in \sigma\}$$

Then  $m \mapsto \gamma_e(m)$  is an isomorphism from  $H(e)$  onto a group of permutations on  $\Gamma$ .

**Proof.** First,  $\gamma_e(m)$  is a map. Indeed, let  $s \xrightarrow{m} t \xrightarrow{m^{-1}} s$  and  $s' \xrightarrow{m} t' \xrightarrow{m^{-1}} s'$ . If  $s \xrightarrow{e} s'$ , we have  $t \xrightarrow{m^{-1}} s \xrightarrow{e} s' \xrightarrow{m} t'$  and thus  $t \xrightarrow{e} t'$ . By a symmetrical proof, we obtain that  $\gamma_e(m)$  is a permutation.

Next, it is easy to verify that  $\gamma_e$  is a morphism.

Finally,  $\gamma_e$  is injective. Indeed, assume that for  $m, m' \in H(e)$  we have  $\gamma_e(m) = \gamma_e(m')$ . Suppose that  $p \xrightarrow{m} q$ .

Assume first that  $p$  is a fixed point of  $e$ . Let  $r, r'$  be such that  $p \xrightarrow{m} r \xrightarrow{m^{-1}} p$  and  $p \xrightarrow{m'} r' \xrightarrow{m'^{-1}} p$ . Since  $\gamma_e(m) = \gamma_e(m')$ , we obtain that  $r, r'$  are in the same element of  $\Gamma$ . We conclude that  $p \xrightarrow{m'} r' \xrightarrow{e} r \xrightarrow{m^{-1}} p \xrightarrow{m} q$  which implies that  $p \xrightarrow{m'} q$ .

Now if  $p$  is not a fixed point of  $e$ , since  $em = m$ , there is an  $r$  such that  $p \xrightarrow{e} r \xrightarrow{m} q$ . By Proposition 14, there is a fixed point  $r'$  of  $e$  such that  $p \xrightarrow{e} r' \xrightarrow{e} r \xrightarrow{m} q$ . Then  $r' \xrightarrow{m} q$  implies  $r' \xrightarrow{m'} q$  by the preceding argument, and finally  $p \xrightarrow{m'} q$ . ◀

We denote  $G_e = \gamma_e(H(e))$ . The definition of  $\gamma_e$  can be formulated differently.

► **Proposition 17.** *Let  $M$  be a monoid of relations on a finite set  $Q$  and let  $e \in M$  be an idempotent. Let  $\sigma, \tau$  be two distinct connected components of fixed points of  $e$  and let  $s \in \sigma, t \in \tau$ . If  $e_{s,t} = 1$ , then  $m_{t,s} = 0$  for every  $m \in H(e)$  and thus  $(\sigma, \tau) \notin \gamma_e(m)$ . If  $e_{s,t} = e_{t,s} = 0$  then  $s \xrightarrow{m} t$  implies  $(\sigma, \tau) \in \gamma_e(m)$ .*

**Proof.** Assume first that  $e_{s,t} = 1$  so that the restriction of  $e$  to  $\{s, t\}$  is the matrix  $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ . If  $m_{t,s} = 1$ , then the restriction of  $m$  to  $\{s, t\}$  is the matrix with all ones, which is impossible since no power of  $m$  can be equal to  $e$ . If the restriction of  $e$  to  $\{s, t\}$  is the identity, then the restriction of  $m \in H(e)$  is a permutation. Thus  $(\sigma, \tau) \in \gamma_e(m)$  if and only if  $s \xrightarrow{m} t$ . ◀

The following extends Proposition 9.1.9 in [3]. It uses the Green relation  $\mathcal{D} = \mathcal{LR} = \mathcal{RL}$ . Two permutation groups  $G$  over  $Q$  and  $G'$  over  $Q'$  are called *equivalent* if there exists a bijection  $\alpha : Q \rightarrow Q'$  and an isomorphism  $\psi : G \rightarrow G'$  such that for all  $q \in Q$  and  $g \in G$  we have  $\alpha(q.g) = \alpha(q).\psi(g)$ , where  $q.g$  is the action of  $g$  on  $q$  (see Section 1.13 of [3]). In a more standard terminology, two permutation groups are equivalent if and only if their group actions are isomorphic, though we use the terminology of [3] to simplify the comparison with the results described there.

► **Proposition 18.** *Let  $M$  be a monoid of relations on a finite set  $Q$  and let  $e, e' \in M$  be  $\mathcal{D}$ -equivalent idempotents. Then the groups  $G_e$  and  $G_{e'}$  are equivalent permutation groups.*

**Proof.** Let  $(a, a', b, b')$  be a passing system from  $e$  to  $e'$ , that is such that

$$eaa' = e, \quad bb'e' = e', \quad ea = b'e'.$$

We will verify that there is a commutative diagram of isomorphisms shown in Figure 7.

$$\begin{array}{ccc}
 H(e) & \xrightarrow{\tau} & H(e') \\
 \downarrow \gamma_e & & \downarrow \gamma_{e'} \\
 G_e & \xrightarrow{\tau'} & G_{e'}
 \end{array}$$

■ **Figure 7** Commutative diagram of isomorphisms.

We define the map  $\tau$  by  $\tau(m) = bma$ . Then it is easy to verify that  $\tau$  is a morphism and that  $m' \mapsto b'm'a'$  is its inverse. Thus  $\tau$  is an isomorphism.

We define  $\tau'$  as follows. Let  $\Gamma_e, \Gamma_{e'}$  be the sets of strongly connected components of fixed points of  $e$  and  $e'$  respectively. Let  $\theta$  be the relation between  $\Gamma_e$  and  $\Gamma_{e'}$  defined by  $(\sigma, \sigma') \in \theta$  if for some  $s \in \sigma$  and  $s' \in \sigma'$ , we have  $s \xrightarrow{eae'} s'$ . One may verify that  $\theta$  is a bijection between  $\Gamma_e$  and  $\Gamma_{e'}$ . Its inverse is the map on classes induced by  $e'be = e'a'e$ . Then  $\tau'(n) = \theta^t n \theta$ .

We verify that the diagram is commutative. Suppose that for some  $m \in H(e)$  ( $\sigma'_1, \sigma'_1 \in \tau'(\gamma_e(m))$ ). By definition of  $\tau'$  there exist  $\sigma_1, \sigma_2 \in \Gamma_e$  such that

$$(\sigma'_1, \sigma_1) \in \theta^t, \quad (\sigma_1, \sigma_2) \in \gamma_e(m) \text{ and } (\sigma_2, \sigma'_2) \in \theta.$$

Then for  $s_1 \in \sigma_1, s'_1 \in \sigma'_1, s'_2 \in \sigma'_2$  and  $s_2 \in \sigma_2$ , we have

$$s'_1 \xrightarrow{e'be} s_1, \quad s_1 \xrightarrow{m} s_2 \xrightarrow{m^{-1}} s_1, \quad s_2 \xrightarrow{eae'} s'_2.$$

Then  $s'_1 \xrightarrow{bma} s'_2 \xrightarrow{bm^{-1}a} s'_1$  showing that  $(\sigma'_1, \sigma'_1) \in \gamma_{e'}(\tau(m))$ . ◀

Note that, contrary to the case of a monoid of unambiguous relations, two  $\mathcal{D}$ -equivalent idempotents need not have the same number of fixed points, as shown by the following example.

► **Example 19.** Let  $M$  be the monoid of all relations on  $Q = \{1, 2\}$ . The two idempotents

$$e = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad e' = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

are  $\mathcal{D}$ -equivalent although the first has one fixed point and the second has two.

Let  $M$  be a monoid of relations on a finite set  $Q$ . The *minimal rank* of  $M$ , denoted  $r(M)$  is the minimum of the ranks of the elements of  $M$  other than 0. The following statement generalizes Theorem 9.3.10 in [3] from unambiguous to arbitrary transitive monoids of relations. A  $\mathcal{D}$ -class is *regular* if it contains an idempotent. A monoid of relations on  $Q$  is *transitive* if for every  $p, q \in Q$ , there is an  $m \in M$  such that  $p \xrightarrow{m} q$ .

► **Theorem 20.** *Let  $M$  be a transitive monoid of relations on a finite set  $Q$ . The set  $K$  of elements of rank  $r(M)$  is a regular  $\mathcal{D}$ -class. The groups  $G_e$  for  $e$  idempotent in  $K$  are equivalent transitive permutation groups. Moreover, for a fixed point  $i$  of  $e$ , the minimal rank  $r(M)$  is the index of the subgroup  $\{m \in H(e) \mid i \xrightarrow{m} i\}$  in  $H(e)$ .*

**Proof.** The proof is the same as for the case of an unambiguous monoid of relations except for the last statement. Let  $\sigma, \tau$  be two distinct strongly connected components of fixed points of  $e$  and let  $s \in \sigma, t \in \tau$ . Since  $M$  is transitive there is an  $m \in M$  such that  $s \xrightarrow{m} t$ . Then  $eme$  is not 0 and thus  $eme \in H(e)$ . Similarly, if  $n \in M$  is such that  $t \xrightarrow{n} s$ , then  $ene \in H(e)$ . This implies by Proposition 17 that the restriction of  $e$  to  $\{s, t\}$  is the identity and that  $(\sigma, \tau) \in \gamma_e(eme)$ . Thus  $G_e$  is transitive. The last statement follows from the fact that for any transitive permutation group on a set  $S$ , the number of elements of  $S$  is equal to the index of the subgroup fixing one of the points of  $S$  (Proposition 1.13.2 of [3]). ◀

The *Suschkewitch group* of  $M$  is one of the equivalent groups  $G_e$  for  $e$  of rank  $r(M)$ .

## 6 Group and degree of a set

Let  $\mathcal{A} = (P, i, i)$  and  $\mathcal{B} = (Q, j, j)$  be automata and let  $\rho : P \rightarrow Q$  be a reduction. For  $m = \varphi_{\mathcal{A}}(w)$ , the relation  $n = \varphi_{\mathcal{B}}(w)$  is well defined. We denote it by  $n = \hat{\rho}(m)$ . Then  $\hat{\rho}$  is a morphism from  $\varphi_{\mathcal{A}}(A^*)$  onto  $\varphi_{\mathcal{B}}(A^*)$  called the *morphism associated with  $\rho$* . The following result extends Proposition 9.5.1 in [3] to arbitrary finite sets of words.

► **Proposition 21.** *Let  $X \subset A^+$  be finite. Let  $\mathcal{A} = (P, i, i)$  and  $\mathcal{B} = (Q, j, j)$  be trim automata recognizing  $X^*$  with multiplicities. Let  $M = \varphi_{\mathcal{A}}(A^*)$  and  $N = \varphi_{\mathcal{B}}(A^*)$ . Let  $E$  be the set of idempotents in  $M$  and  $F$  the set of idempotents in  $N$ .*

*Let  $\rho$  be a sharp reduction of  $\mathcal{A}$  onto  $\mathcal{B}$  and let  $\hat{\rho} : M \rightarrow N$  be the morphism associated with  $\rho$ . Then*

1.  $\hat{\rho}(E) = F$ .
2. *Let  $e \in E$  and  $f = \hat{\rho}(e)$ . The restriction of  $\rho$  to the set  $S$  of fixed points of  $e$  is a bijection on the set of fixed points of  $f$ , and the groups  $H_e$  and  $H_f$  are equivalent.*

**Proof.** 1. Let  $e \in E$ . Then  $\hat{\rho}(e)$  is idempotent since  $\hat{\rho}$  is a morphism. Thus  $\hat{\rho}(E) \subset F$ . Conversely, if  $f \in F$ , let  $w \in A^*$  be such that  $\varphi_{\mathcal{B}}(w) = f$ . Let  $n \geq 1$  be such that  $e = \varphi_{\mathcal{A}}(w)^n$  is idempotent. Then  $\hat{\rho}(e) = f$  since  $\hat{\rho} \circ \varphi_{\mathcal{A}} = \varphi_{\mathcal{B}}$ .

2. Let  $S$  be the set of fixed points of  $e$  and  $T$  the set of fixed points of  $f$ . Consider  $s \in S$  and let  $t = \rho(s)$ . From  $s \xrightarrow{e} s$ , we obtain  $t \xrightarrow{f} t$  and thus  $\rho(S) \subset T$ . Conversely, let  $t \in T$ . The restriction of  $e$  to the set  $R = \rho^{-1}(t)$  is a non zero idempotent. Thus there is some  $s \in R$  which is a fixed point of this idempotent, and thus of  $e$ . Thus  $t \in \rho(S)$ .

Since  $\hat{\rho}$  is a morphism from  $M$  onto  $N$ , we have  $\hat{\rho}(H(e)) = H(f)$ . It is clear that  $\rho$  maps a strongly connected component of  $e$  on a strongly connected component of  $f$ . To show that this map is a bijection, consider  $s, s' \in S$  such that  $\rho(s), \rho(s')$  belong to the same connected component. We may assume that  $e$  is not the equality relation. Let  $w \in A^+$  be such that  $\varphi_{\mathcal{A}}(w) = e$ . Since  $X$  is finite, there are factorizations  $w = uv = u'v'$  such that  $s \xrightarrow{u} i \xrightarrow{v} s$  and  $s' \xrightarrow{u'} i \xrightarrow{v'} s'$ . Then we have  $j \xrightarrow{v} \rho(s) \xrightarrow{w} \rho(s') \xrightarrow{u'} j$ . Since  $\rho$  is sharp, this implies  $i \xrightarrow{v w u'} i$  and finally  $s \xrightarrow{u v w u' v'} s'$ . This shows that  $s \xrightarrow{e} s'$ . A similar proof shows that  $s' \xrightarrow{e} s$ . Thus,  $s, s'$  belong to the same connected component of  $e$ .

Moreover, for every  $m \in H(e)$ , one has  $s \xrightarrow{m} t \xrightarrow{m^{-1}} s$  if and only if  $\rho(s) \xrightarrow{\hat{\rho}(m)} \rho(t) \xrightarrow{\hat{\rho}(m^{-1})} \rho(s)$ . Thus  $H_e$  and  $H_f$  are equivalent permutation groups. ◀

Let  $X \subset A^+$  be a finite set and let  $\mathcal{A}$  be the flower automaton of  $X$ . The *degree* of  $X$ , denoted  $d(X)$  is the minimal rank of the monoid  $M = \varphi_{\mathcal{A}}(A^*)$ . The *group of  $X$*  is the Suschkevitch group of  $M$ . Proposition 21 shows that the definitions of the group and of the degree do not depend on the automaton chosen to recognize  $X^*$ , provided one takes a trim automaton recognizing  $X^*$  with multiplicities.

## 7 Synchronization

Let  $X \subset A^+$  be a finite set of words. A word  $x \in A^*$  is *synchronizing* for  $X$  if for every  $u, v \in A^*$ ,  $uxv \in X^* \Rightarrow ux, xv \in X^*$ . A set  $X$  is *synchronizing* if there is a synchronizing word  $x \in X^*$ . The next proposition generalizes Proposition 10.1.11 of [3]

► **Proposition 22.** *A finite set  $X \subset A^+$  is synchronizing if and only if its degree  $d(X)$  is 1.*

**Proof.** Let  $\mathcal{A} = (Q, i, i)$  be a trim automaton recognizing  $X^*$ .

Assume first that  $d(X) = 1$ . Let  $x \in X^*$  be such that  $\varphi_{\mathcal{A}}(x)$  has rank 1. If  $uxv \in X^*$ , we have  $i \xrightarrow{u} p \xrightarrow{x} q \xrightarrow{v} i$  for some  $p, q \in Q$ . Since  $\varphi_{\mathcal{A}}(x)$  has rank 1, we deduce from  $i \xrightarrow{x} i$  and  $p \xrightarrow{x} q$  that we have also  $i \xrightarrow{x} q$  and  $p \xrightarrow{x} i$ . Thus  $ux, xv \in X^*$ , showing that  $x$  is synchronizing.

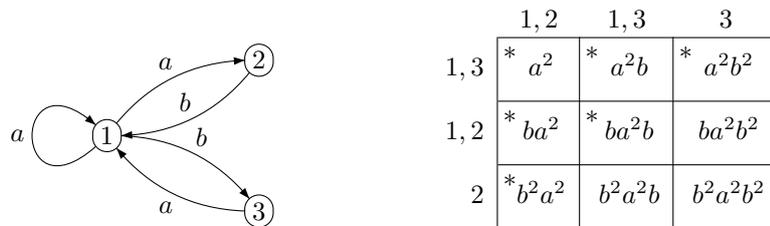
Assume conversely that  $X$  is synchronizing. Let  $x \in X^*$  be a synchronizing word. Replacing  $x$  by some its power, we may assume that  $\varphi_{\mathcal{A}}(x)$  is an idempotent  $e$ . Let  $m \in H(e)$  and let  $w \in \varphi_{\mathcal{A}}^{-1}(m)$ . Since  $H(e)$  is finite, there is some  $n \geq 1$  such that  $m^n = e$ . Then  $(me)^n = e$  implies that  $(wx)^n \in X^*$ . Since  $x$  is synchronizing, we obtain  $wx \in X^*$  and since  $\varphi_{\mathcal{A}}(wx) = me = m$ , this implies  $w \in X^*$ . This shows that  $H(e)$  is contained in  $\varphi_{\mathcal{A}}(X^*)$  and thus that  $d(X) = 1$  by Theorem 20. ◀

► **Example 23.** Consider again  $X = \{a, ab, ba\}$  (Example 3). The flower automaton of  $X$  is represented again for convenience in Figure 8 (left).

The minimal rank of the elements of  $\varphi_{\mathcal{A}}(A^*)$  is 1. Indeed, we have

$$\varphi_{\mathcal{A}}(a^2) = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

Accordingly,  $aa$  is a synchronizing word.



■ **Figure 8** The flower automaton of  $X$  (left) and the set  $K$  of elements of rank 1 (right).

The set  $K$  of elements of rank 1 is represented in Figure 8 (right). For each  $\mathcal{H}$ -class, we indicate on its left the set of states  $p$  such that the row of index  $p$  is nonzero. Similarly, we indicate above it the set of states  $q$  such that the column of index  $q$  is nonzero. A star \* indicates an  $\mathcal{H}$ -class which is a group. Note that

$$\varphi_{\mathcal{A}}(a^2b) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

has two fixed points but only one strongly connected class, in agreement with fact that it is of rank 1.

## 8 Groups and composition

Given a transitive permutation group  $G$  on a set  $Q$ , an *imprimitivity relation* of  $G$  is an equivalence on  $Q$  compatible with the group action. If  $\theta$  is such an equivalence relation, we denote by  $G_\theta$  the permutation group induced by the action of  $G$  on the classes of  $\theta$ . The groups induced by the action on the class of an element  $i \in Q$  by the action of the elements of  $G$  stabilizing the class of  $i$  are all equivalent. We denote by  $G^\theta$  one of them. For two permutation groups  $G, H$  on sets  $P$  and  $Q$  respectively, we denote  $G \leq H$  if there is an imprimitivity equivalence  $\theta$  on  $Q$  such that  $G = H_\theta$ .

The next theorem generalizes Proposition 11.1.2 of [3].

## 54:14 The Degree of a Finite Set of Words

► **Theorem 24.** *Let  $X \subset A^+$  be a finite set with a trim decomposition  $X = Y \circ Z$ , where  $Y$  is complete. There exists an imprimitivity equivalence  $\theta$  of  $G = G(X)$  such that*

$$G^\theta \leq G(Y), \quad G_\theta = G(Z).$$

*In particular,  $d(X) \leq d(Y) \cdot d(Z)$ .*

**Proof.** Let  $\mathcal{B} = (Q, i, i)$  be the flower automaton of  $Y$  and let  $\mathcal{T}$  be the prefix transducer of  $Z$ . Let  $\mathcal{A} = \mathcal{B} \circ \mathcal{T}$ . By Proposition 12, there is a reduction  $\rho$  from  $\mathcal{A} = (Q \times P, (i, 1), (i, 1))$  onto the prefix automaton  $\mathcal{C}$  of  $Z$ .

Let  $e$  be an idempotent of minimal rank in  $\varphi_{\mathcal{A}}(X^*)$ . Let  $S$  be the set of fixed points of  $e$  and let  $\Gamma$  be the set of connected components (scc) of the elements of  $S$ . Let  $\hat{S}$  be the set of fixed points of  $\hat{e} = \hat{\rho}(e)$  and let  $\hat{\Gamma}$  be the set of corresponding scc's. If  $s, s' \in S$  are in the same scc, then  $\rho(s), \rho(s')$  are in the same scc of  $\hat{S}$ . Thus, we have a well-defined map  $\bar{\rho}: \Gamma \rightarrow \hat{\Gamma}$  such that  $s \in \Gamma$  if and only if  $\rho(s) \in \bar{\rho}(\Gamma)$ .

We define an equivalence  $\theta$  on  $\Gamma$  by  $\sigma \equiv \sigma'$  if  $\bar{\rho}(\sigma) = \bar{\rho}(\sigma')$ . Let  $m \in H(e)$  and suppose that  $(\sigma, \tau), (\sigma', \tau') \in \gamma_e(m)$ . If  $\sigma \equiv \sigma' \pmod{\theta}$ , then  $\tau \equiv \tau' \pmod{\theta}$ . Let indeed  $s \in \sigma, s' \in \sigma'$  and  $t \in \tau, t' \in \tau'$ . We have by definition of  $\gamma_e$

$$s \xrightarrow{m} t \xrightarrow{m^{-1}} s \text{ and } s' \xrightarrow{m} t' \xrightarrow{m^{-1}} s'$$

and thus

$$\rho(s) \xrightarrow{\hat{\rho}(m)} \rho(t) \xrightarrow{\hat{\rho}(m)^{-1}} \rho(s) \text{ and } \rho(s') \xrightarrow{\hat{\rho}(m)} \rho(t') \xrightarrow{\hat{\rho}(m)^{-1}} \rho(s')$$

This implies that  $\rho(t) \xrightarrow{\hat{e}} \rho(t')$  and  $\rho(t') \xrightarrow{\hat{e}} \rho(t)$ . But since  $\gamma_e(\hat{m})$  is a permutation, this forces  $\bar{\rho}(\tau) = \bar{\rho}(\tau')$  and finally  $\tau \equiv \tau' \pmod{\theta}$ . Since the action of  $H(e)$  on the classes of  $\theta$  is the same as the action of  $H(\hat{e})$ , we have  $G(Z) = G_\theta$ .

Finally, let  $\sigma \in \Gamma$  be the scc of the initial state  $(i, 1)$  and let  $I$  be its class mod  $\theta$ . Thus  $d(X) = \text{Card}(I)d(Z)$ . Let  $x \in X^*$  be such that  $\varphi_{\mathcal{A}}(x) = e$  and let  $y = \beta^{-1}(x)$ . Then  $f = \varphi_{\mathcal{B}}(y)$  is an idempotent of  $\varphi_{\mathcal{B}}(B^*)$  of rank  $d(Y)$ . Let  $U$  be the set of fixed points of  $f$  and let  $\Phi$  be the set of scc of  $U$  for the action of  $f$ . Let  $\sigma$  be the equivalence on  $\Phi$  induced by the equivalence  $r \equiv s$  if  $(r, 1), (s, 1)$  belong to the same scc for  $e$ . Then  $\sigma$  is an imprimitivity equivalence for  $G(Y)$  such that  $G(Y)_\sigma = G^\theta$ . Thus  $G^\theta \leq G(Y)$  and  $\text{Card}(I) \leq d(Y)$ , which implies  $d(X) \leq d(Y) \cdot d(Z)$ . ◀

► **Example 25.** Let  $Z = \{a, ab, ba, ca\}$  and  $X = Z^2$ . We have  $X = Y \circ_\beta Z$  with  $Y = \{u, v, w, x\}^2$  and  $\beta: u \mapsto a, v \mapsto ab, w \mapsto bc, x \mapsto ca$ . The word  $aa$  is synchronizing for  $Z$  and thus  $d(Z) = 1$ . In contrast, we have  $d(Y) = 2$  and  $G(Y) = \mathbb{Z}/2\mathbb{Z}$ . It can be verified that the word  $ca^2b$  is synchronizing for  $X$  and thus  $d(X) = 1$ . Thus  $d(X) < d(Y) \cdot d(Z) = 2 \cdot 1 = 2$ . Thus the case of a strict inequality can occur. This is made possible by the fact that  $Z$  is not a code. Indeed, we have  $(ab)(ca) = a(bc)a$ .

## 9 Decompositions of codes

Finally, we use the developed techniques to show that for a uniquely decipherable code  $X$  for all the trim decompositions of the form  $X = Y \circ Z$  with  $Y$  complete we have that  $Z$  (and thus  $Y$ ) is a uniquely decipherable code as well. It shows that, as long as we require  $Y$  to be complete, we do not get any new trim decompositions of uniquely decipherable codes even if we decompose them as arbitrary sets of words.

► **Proposition 26.** *Let  $X = Y \circ Z$  be a trim decomposition of a finite set  $X$ . If  $X$  is a uniquely decipherable code and if  $Y$  is complete, then  $Z$  is a uniquely decipherable code.*

**Proof.** Since  $\beta$  is trim,  $Y$  is a uniquely decipherable code. Let  $\beta : B \rightarrow Z$  be the coding morphism for  $Z$  such that  $X = Y \circ_{\beta} Z$ . Assume that  $z \in Z^*$  is a word with more than one factorization into words of  $Z$ . Let  $u, v \in B^*$  two distinct elements in  $\beta^{-1}(z)$ . Let  $\mathcal{A}$  be the flower automaton of  $Y$ . Let  $y \in Y^*$  be such that  $\varphi_{\mathcal{A}}(y)$  has minimal rank. Then  $yuy, yvy$  are not zero since  $Y$  is complete. Thus  $\varphi_{\mathcal{A}}(yuy), \varphi_{\mathcal{A}}(yvy)$  belong to the  $\mathcal{H}$ -class of  $\varphi_{\mathcal{A}}(y)$  which is a finite group. Let  $e$  be its idempotent. There are integers  $n, m, p$  such that  $\varphi_{\mathcal{A}}(y)^n = \varphi_{\mathcal{A}}(yuy)^m = \varphi_{\mathcal{A}}(yvy)^p = e$ . Since  $y \in Y^*$ , this implies that  $e \in \varphi_{\mathcal{A}}(Y^*)$  and thus that  $(yuy)^m, (yvy)^p$  are in  $Y^*$ . We conclude that  $Y$  is not a uniquely decipherable code, a contradiction. ◀

This is false if we do not require  $Y$  to be complete. Consider a code  $X = \{ab, abaab, abbab\}$ , which can be decomposed into  $X = Y \circ Z$  with  $Y = \{u, uvu, uvu\}$  and  $Z = \{ab, a, b\}$ . The decomposition is obviously trim, the set  $X$  is a uniquely decipherable code, but the set  $Z$  is not a uniquely decipherable code.

---

## References

- 1 Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text compression*. Prentice-Hall, Inc., 1990.
- 2 Jean Berstel, Dominique Perrin, Jean-Francois Perrot, and Antonio Restivo. Sur le théorème du défaut. *Journal of Algebra*, 60(1):169–180, 1979. doi:10.1016/0021-8693(79)90113-3.
- 3 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Cambridge University Press, 2009.
- 4 Renato M. Capocelli, Luisa Gargano, and Ugo Vaccaro. A fast algorithm for the unique decipherability of multivalued encodings. *Theoretical Computer Science*, 134(1):63–78, 1994. doi:10.1016/0304-3975(94)90278-X.
- 5 Arturo Carpi and Flavio D’Alessandro. On incomplete and synchronizing finite sets. *Theor. Comput. Sci.*, 664:67–77, 2017. doi:10.1016/j.tcs.2015.08.042.
- 6 Julien Clément, Jean-Pierre Duval, Giovanna Guaiana, Dominique Perrin, and Giuseppina Rindone. Parsing with a finite dictionary. *Theoretical Computer Science*, 340(2):432–442, 2005. doi:10.1016/j.tcs.2005.03.030.
- 7 Aldo de Luca, Dominique Perrin, Antonio Restivo, and Settimo Termini. Synchronization and simplification. *Discrete Mathematics*, 27(3):297–308, 1979. doi:10.1016/0012-365X(79)90164-X.
- 8 Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.
- 9 Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.
- 10 Fernando Guzmán. Decipherability of codes. *Journal of Pure and Applied Algebra*, 141(1):13–35, 1999. doi:10.1016/S0022-4049(98)00019-X.
- 11 Tero Harju and Juhani Karhumäki. Many aspects of defect theorems. *Theoretical Computer Science*, 324(1):35–54, 2004. Words, Languages and Combinatorics. doi:10.1016/j.tcs.2004.03.051.
- 12 Evelyne Le Rest and Michel Le Rest. Une représentation fidèle des groupes d’un monoïde de relations binaires sur un ensemble fini. *Semigroup Forum*, 21(2-3):167–172, 1980. doi:10.1007/BF02572547.
- 13 Abraham Lempel. On multiset decipherable codes (corresp.). *IEEE Trans. Inf. Theor.*, 32(5):714–716, 1986. doi:10.1109/TIT.1986.1057217.
- 14 J.S. Montague and R.J. Plemmons. Maximal subgroups of the semigroup of relations. *Journal of Algebra*, 13(4):575–587, 1969.

## 54:16 The Degree of a Finite Set of Words

- 15 H. Nagumo, Mi Lu, and Karan Watson. Parallel algorithms for the static dictionary compression. In *Proceedings DCC '95 Data Compression Conference*, pages 162–171, 1995.
- 16 R. J. Plemmons and M. T. West. On the semigroup of binary relations. *Pacific J. Math.*, 35(3):743–753, 1970.
- 17 Evelyne Barbin-Le Rest and Stuart W. Margolis. On the group complexity of a finite language. In *Proceedings of the 10th Colloquium on Automata, Languages and Programming*, pages 433–444, Berlin, Heidelberg, 1983. Springer-Verlag.
- 18 Antonio Restivo. A note on multiset decipherable codes. *IEEE Trans. Inf. Theor.*, 35(3):662–663, 1989. doi:10.1109/18.30991.
- 19 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
- 20 Marcel-Paul Schützenberger. A property of finitely generated submonoids of free monoids. In G. Pollak, editor, *Algebraic Theory of Semigroups*, pages 545–576. North-Holland, 1979.
- 21 Andreas Weber and Tom Head. The finest homophonic partition and related code concepts. *IEEE Transactions on Information Theory*, 42(5):1569–1575, 1996.

# What You Must Remember When Transforming Datawords

M. Praveen

Chennai Mathematical Institute, India  
UMI ReLaX, Indo-French joint research unit

---

## Abstract

Streaming Data String Transducers (SDSTs) were introduced to model a class of imperative and a class of functional programs, manipulating lists of data items. These can be used to write commonly used routines such as insert, delete and reverse. SDSTs can handle data values from a potentially infinite data domain. The model of Streaming String Transducers (SSTs) is the fragment of SDSTs where the infinite data domain is dropped and only finite alphabets are considered. SSTs have been much studied from a language theoretical point of view. We introduce data back into SSTs, just like data was introduced to finite state automata to get register automata. The result is Streaming String Register Transducers (SSRTs), which is a subclass of SDSTs.

We use origin semantics for SSRTs and give a machine independent characterization, along the lines of Myhill-Nerode theorem. Machine independent characterizations for similar models are the basis of learning algorithms and enable us to understand fragments of the models. Origin semantics of transducers track which positions of the output originate from which positions of the input. Although a restriction, using origin semantics is well justified and is known to simplify many problems related to transducers. We use origin semantics as a technical building block, in addition to characterizations of deterministic register automata. However, we need to build more on top of these to overcome some challenges unique to SSRTs.

**2012 ACM Subject Classification** Theory of computation → Transducers; Theory of computation → Automata over infinite objects

**Keywords and phrases** Streaming String Transducers, Data words, Machine independent characterization

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.55

**Related Version** A full version of the extended abstract is available at <https://arxiv.org/abs/2005.02596>.

**Funding** *M. Praveen*: Partially supported by a grant from the Infosys foundation.

**Acknowledgements** The author thanks C. Aiswarya, Kamal Lodaya, K. Narayan Kumar and anonymous reviewers for suggestions to improve the presentation and pointers to related works.

## 1 Introduction

Transductions are in general relations among words. Transducers are theoretical models that implement transductions. Transducers are used in a variety of applications, such as analysis of web sanitization frameworks, host based intrusion detection, natural language processing, modeling some classes of programming languages and constructing programming language tools like evaluators, type checkers and translators. Streaming Data String Transducers (SDSTs) were introduced in [2] to model a class of imperative and a class of functional programs, manipulating lists of data items. Transducers have been used in [16] to infer semantic interfaces of data structures such as stacks. Such applications use Angluin style learning, which involves constructing transducers by looking at example operations of the object under study. Since the transducer is still under construction, we need to make inferences about the transduction without having access to a transducer which implements it. Theoretical bases for doing this are machine independent characterizations, which identify what kind of transductions can be implemented by what kind of transducers and give a



© M. Praveen;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 55; pp. 55:1–55:14

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

template for constructing transducers. Indeed the seminal Myhill-Nerode theorem gives a machine independent characterization for regular languages over finite alphabets, which form the basis of Angluin style learning of regular languages [3]. A similar characterization for a fragment of SDSTs is given in [5] and is used as a basis to design a learning algorithm.

Programs deal with data from an infinite domain and transducers modeling the programs should also treat data as such. For example in [16], the state space reduced from  $10^9$  to 800 and the number of learning queries reduced from billions to 4000 by switching to a transducer model that can deal with data from an infinite domain. We give a machine independent characterization for a fragment of SDSTs more powerful than those in [16, 5]. The additional power comes from significant conceptual differences. The transducers used in [16] produce the output in a linear fashion without remembering what was output before. For example, they cannot output the reverse of the input strings, which can be done by our model. The model studied in [5] are called Streaming String Transducers (SSTs), the fragment obtained from SDSTs by dropping the ability to deal with data values from an infinite domain. We retain this ability in our model, called Streaming String Register Transducers (SSRTs). It is obtained from SDSTs by dropping the ability to deal with linear orders in the data domain. Apart from Angluin style learning algorithms, machine independent characterizations are also useful for studying fragments of transducer models. E.g. in [5], machine independent characterization of SSTs is used to study fragments such as non-deterministic automata with output and transductions definable in First Order logic.

We use origin semantics of transducers, which are used in [5] to take into account how positions of the output originate from the positions of the input. Using origin semantics is known to ease some of the problems related to transducers, e.g., [7]. Origin semantics is a restriction, but a reasonable one and is used extensively in this paper.

## Contributions

Machine independent characterizations are known for automata over data values from an infinite domain [15, 4] and for streaming transducers over finite alphabets [5], but not for streaming transducers over data values, which is what we develop here. This involves both conceptual and technical challenges. In [15, 4], data values that must be remembered by an automaton while reading a word from left to right are identified using a machine independent definition. We lift this to transducers and identify that the concept of factored outputs from [5] is necessary for this. Factored outputs can let us ignore some parts of transduction outputs, which is necessary to define when two words behave similarly. However, [5] does not deal with data values from an infinite domain and it takes quite a bit of manipulation with permutations on data values to make ideas from there work here. In transductions, suffixes can influence how prefixes are transformed. This is elegantly handled in [5] using two way transducer models known to be equivalent to SSTs. There are no such models known when data values are present. To handle it in a one way transducer model, we introduce data structures based on trees that keep track of all possible suffixes. This does raise the question of whether there are interesting two way transducer models with data values. Recent work [6] has made progress in this direction, which we discuss at the end of this extended abstract. We concentrate here on SDSTs and its fragments, which are known to be equivalent to classes of imperative and functional programming languages. In [2], it is explained in detail which features of programming languages correspond to which features of the transducer. Over finite alphabets, streaming string transducers are expressively equivalent to regular transductions, which are also defined by two way deterministic finite-state transducers and by monadic second order logic [1].

## Related Works

Studying transducer models capable of handling data values from an infinite domain is an active area of research [13, 14]. Streaming transducers like SDSTs have the distinctive feature of using variables to store intermediate values while computing transductions; this idea appears in an earlier work [11] that introduced *simple programs on strings*, which implement the same class of transductions as those implemented by SSTs. An Angluin style learning algorithm for deterministic automata with memory is given in [17]. A machine independent characterization of automata with finite memory is given in [8], which is further extended to data domains with arbitrary binary relations in [9]. The learning algorithm of [17] is extended to Mealy machines with data in [16]. However, Mealy machines are not as powerful as SSRTs that we consider here. Using a more abstract approach of nominal automata, [19] presents a learning algorithm for automata over infinite alphabets. Logical characterizations of transducers that can handle data are considered in [12]. However, the transducers in that paper cannot use data values to make decisions, although they are part of the output. Register automata with linear arithmetic introduced in [10] shares some of the features of the transducer model used here. Here, data words stored in variables can be concatenated, while in register automata with linear arithmetic, numbers stored in variables can be operated upon by linear operators.

Most proofs and some technical details in this extended abstract are skipped due to space constraints. All the proofs and technical details can be found in the full version.

## 2 Preliminaries

Let  $\mathbb{I}$  be the set of integers,  $\mathbb{N}$  be the set of non-negative integers and  $D$  be an infinite set of data values. We will refer to  $D$  as the *data domain*. For  $i, j \in \mathbb{I}$ , we denote by  $[i, j]$  the set  $\{k \mid i \leq k \leq j\}$ . For any set  $S$ ,  $S^*$  denotes the set of all finite sequences of elements from  $S$ . The empty sequence is denoted by  $\epsilon$ . Given  $u, v \in S^*$ ,  $v$  is a *prefix* (resp. *suffix*) of  $u$  if there exists  $w \in S^*$  such that  $u = vw$  (resp.  $u = wv$ ). The sequence  $v$  is an *infix* of  $u$  if there are sequences  $w_1, w_2$  such that  $u = w_1vw_2$ .

Let  $\Sigma, \Gamma$  be finite alphabets. We will use  $\Sigma$  for input alphabet and  $\Gamma$  for output alphabet. A *data word* over  $\Sigma$  is a word in  $(\Sigma \times D)^*$ . A *data word with origin information* over  $\Gamma$  is a word in  $(\Gamma \times D \times \mathbb{N})^*$ . Suppose  $\Sigma = \{\text{title, firstName, lastName}\}$  and  $\Gamma = \{\text{givenName, surName}\}$ . An example data word over  $\Sigma$  is  $(\text{title, Mr.})(\text{firstName, Harry})(\text{lastName, Tom})$ . If we were to give this as input to a device that reverses the order of names, the output would be the data word with origin information  $(\text{surName, Tom}, 3)(\text{givenName, Harry}, 2)$ , over  $\Gamma$ . In the triple  $(\text{givenName, Harry}, 2)$ , the third component 2 indicates that the pair  $(\text{givenName, Harry})$  originates from the second position of the input data word. We call the third component *origin* and it indicates the position in the input that is responsible for producing the output triple. If a transduction is being implemented by a transducer, the origin of an output position is the position of the input that the transducer was reading when it produced the output. The data value at some position of the output may come from any position (not necessarily the origin) of the input data word. We write *transduction* for any function from data words over  $\Sigma$  to data words with origin information over  $\Gamma$ .

For a data word  $w$ ,  $|w|$  is its length. For a position  $i \in [1, |w|]$ , we denote by  $\text{data}(w, i)$  (resp.  $\text{letter}(w, i)$ ) the data value (resp. the letter from the finite alphabet) at the  $i^{\text{th}}$  position of  $w$ . We denote by  $\text{data}(w, *)$  the set of all data values that appear in  $w$ . For positions  $i \leq j$ , we denote by  $w[i, j]$  the infix of  $w$  starting at position  $i$  and ending at position  $j$ . Note that  $w[1, |w|] = w$ . Two data words  $w_1, w_2$  are *isomorphic* (denoted by

$w_1 \simeq w_2$ ) if  $|w_1| = |w_2|$ ,  $\mathbf{letter}(w_1, i) = \mathbf{letter}(w_2, i)$  and  $\mathbf{data}(w_1, i) = \mathbf{data}(w_1, j)$  iff  $\mathbf{data}(w_2, i) = \mathbf{data}(w_2, j)$  for all positions  $i, j \in [1, |w_1|]$ . For data values  $d, d'$ , we denote by  $w[d/d']$  the data word obtained from  $w$  by replacing all occurrences of  $d$  by  $d'$ . We say that  $d'$  is a *safe replacement* for  $d$  in  $w$  if  $w[d/d'] \simeq w$ . Intuitively, replacing  $d$  by  $d'$  doesn't introduce new equalities/inequalities among the positions of  $w$ . For example,  $d_1$  is a safe replacement for  $d_2$  in  $(a, d_3)(b, d_2)$ , but not in  $(a, d_1)(b, d_2)$ .

A permutation on data values is any bijection  $\pi : D \rightarrow D$ . For a data word  $u$ ,  $\pi(u)$  is obtained from  $u$  by replacing all its data values by their respective images under  $\pi$ . A transduction  $f$  is *invariant under permutations* if for every data word  $u$  and every permutation  $\pi$ ,  $f(\pi(u)) = \pi(f(u))$  (permutation can be applied before or after the transduction).

Suppose a transduction  $f$  has the property that for any triple  $(\gamma, d, o)$  in any output  $f(w)$ , there is a position  $i \leq o$  in  $w$  such that  $\mathbf{data}(w, i) = d$ . If the data value  $d$  is output from the origin  $o$ , then  $d$  should have already occurred in the input on or before  $o$ . Such transductions are said to be *without data peeking*. We say that a transduction has *linear blow up* if there is a constant  $K$  such that for any position  $o$  of any input, there are at most  $K$  positions in the output whose origin is  $o$ .

### Streaming String Register Transducers

We present an extension of SSTs to handle data values, just like finite state automata were extended to finite memory automata [18]. Our model is a subclass of SDSTs, which can store intermediate values (which can be long words) in variables. E.g., reversing an input word can be achieved as follows: as each input symbol is read, concatenate it to the back of a variable maintained for this purpose. At the end, the variable will have the reverse of the input. There are also registers in these models, which can store single data values. Transitions can be enabled/disabled based on whether the currently read data value is equal/unequal to the one stored in one of the registers.

► **Definition 1.** A Streaming String Register Transducer (*SSRT*) is an eight tuple  $S = (\Sigma, \Gamma, Q, q_0, R, X, O, \Delta)$ , where

- the finite alphabets  $\Sigma, \Gamma$  are used for input, output respectively,
- $Q$  is a finite set of states,  $q_0$  is the initial state,
- $R$  is a finite set of registers and  $X$  is a finite set of data word variables,
- $O : Q \rightarrow ((\Gamma \times \hat{R}) \cup X)^*$  is a partial output function, where  $\hat{R} = R \cup \{\text{curr}\}$ , with *curr* being a special symbol used to denote the current data value being read and
- $\Delta \subseteq (Q \times \Sigma \times \Phi \times Q \times 2^R \times U)$  is a finite set of transitions. The set  $\Phi$  consists of all Boolean combinations of atomic constraints of the form  $r^-$  or  $r^\neq$  for  $r \in R$ . The set  $U$  is the set of all functions from the set  $X$  of data word variables to  $((\Gamma \times \hat{R}) \cup X)^*$ .

It is required that

- For every  $q \in Q$  and  $x \in X$ , there is at most one occurrence of  $x$  in  $O(q)$  and
- for every transition  $(q, \sigma, \phi, q', R', ud)$  and for every  $x \in X$ ,  $x$  appears at most once in the set  $\{ud(y) \mid y \in X\}$ .

We say that the last two conditions above enforce a SSRT to be *copyless*, since it prevents multiple copies of contents being made.

A *valuation*  $val$  for a transducer  $S$  is a partial function over registers and data word variables such that for every register  $r \in R$ , either  $val(r)$  is undefined or is a data value in  $D$ , and for every data word variable  $x \in X$ ,  $val(x)$  is a data word with origin information over  $\Gamma$ . The valuation  $val$  and data value  $d$  satisfies the atomic constraint  $r^-$  (resp.  $r^\neq$ ) if  $val(r)$  is defined and  $d = val(r)$  (resp. undefined or  $d \neq val(r)$ ). Satisfaction is extended to

Boolean combinations in the standard way. We say that a SSRT is *deterministic* if for every two transitions  $(q, \sigma, \phi, q', R', u)$  and  $(q, \sigma, \phi', q'', R'', u')$  with the same source state  $q$  and input symbol  $\sigma$ , the formulas  $\phi$  and  $\phi'$  are mutually exclusive (i.e.,  $\phi \wedge \phi'$  is unsatisfiable). We consider only deterministic SSRTs here.

A configuration is a triple  $(q, val, i)$  where  $q \in Q$  is a state,  $val$  is a valuation and  $i$  is the number of symbols read so far. The transducer starts in the configuration  $(q_0, val_\epsilon, 0)$  where  $q_0$  is the initial state and  $val_\epsilon$  is the valuation such that  $val_\epsilon(r)$  is undefined for every register  $r \in R$  and  $val_\epsilon(x) = \epsilon$  for every data word variable  $x \in X$ . From a configuration  $(q, val, i)$ , the transducer can read a pair  $(\sigma, d) \in \Sigma \times D$  and go to the configuration  $(q', val', i + 1)$  if there is a transition  $(q, \sigma, \phi, q', R', ud)$  and 1)  $d$  and  $val$  satisfies  $\phi$  and 2)  $val'$  is obtained from  $val$  by assigning  $d$  to all the registers in  $R'$  and for every  $x \in X$ , setting  $val'(x)$  to  $ud(x)[y \mapsto val(y), (\gamma, curr) \mapsto (\gamma, d, i + 1), (\gamma, r) \mapsto (\gamma, val(r), i + 1)]$  (in  $ud(x)$ , replace every occurrence of  $y$  by  $val(y)$  for every data word variable  $y \in X$ , replace every occurrence of  $(\gamma, curr)$  by  $(\gamma, d, i + 1)$  for every output letter  $\gamma \in \Gamma$  and replace every occurrence of  $(\gamma, r)$  by  $(\gamma, val(r), i + 1)$  for every output letter  $\gamma \in \Gamma$  and every register  $r \in R$ ). After reading a data word  $w$ , if the transducer reaches some configuration  $(q, val, n)$  and  $O(q)$  is not defined, then the transducer's output  $\llbracket S \rrbracket(w)$  is undefined for the input  $w$ . Otherwise, the transducer's output is defined as  $\llbracket S \rrbracket(w) = O(q)[y \mapsto val(y), (\gamma, curr) \mapsto (\gamma, d, n), (\gamma, r) \mapsto (\gamma, val(r), n)]$ , where  $d$  is the last data value in  $w$ .

Intuitively, the transition  $(q, \sigma, \phi, q', R', ud)$  checks that the current valuation  $val$  and the data value  $d$  being read satisfies  $\phi$ , goes to the state  $q'$ , stores  $d$  into the registers in  $R'$  and updates data word variables according to the update function  $ud$ . The condition that  $x$  appears at most once in the set  $\{ud(y) \mid y \in X\}$  ensures that the contents of any data word variable are not duplicated into more than one variable. This ensures, among other things, that the length of the output is linear in the length of the input. The condition that for every two transitions  $(q, \sigma, \phi, q', R', ud)$  and  $(q, \sigma, \phi', q'', R'', ud')$  with the same source state and input symbol, the formulas  $\phi$  and  $\phi'$  are mutually exclusive ensures that the transducer cannot reach multiple configurations after reading a data word (i.e., the transducer is deterministic).

► **Example 2.** Consider the transduction that is the identity on inputs in which the first and last data values are equal. On the remaining inputs, the output is the reverse of the input. This can be implemented by a SSRT using two data word variables. As each input symbol is read, it is appended to the front of the first variable and to the back of the second variable. The first variable stores the input and the second one stores the reverse. At the end, either the first or the second variable is output, depending on whether the last data value is equal or unequal to the first data value (which is stored in a register).

In Section 3, we define an equivalence relation on data words and state our main result in terms of the finiteness of the index of the equivalence relation and a few other properties. In Section 4, we prove that transductions satisfying certain properties can be implemented by SSRTs (the backward direction of the main result) and we prove the converse in Section 5.

### 3 How Prefixes and Suffixes Influence Each Other

As is usual in many machine independent characterizations (like the classic Myhill-Nerode theorem for regular languages), we define an equivalence relation on the set of data words to identify similar ones. If the equivalence relation has finite index, it can be used to construct finite state models. We start by looking at what “similar data words” mean in the context of transductions.

Suppose  $L$  is the set of all even length words over some finite alphabet. The words  $a$  and  $aaa$  do the same thing to any suffix  $v$ :  $a \cdot v \in L$  iff  $aaa \cdot v \in L$ . So,  $a$  and  $aaa$  are identified to be similar with respect to  $L$  in the classic machine independent characterization. Instead of a language  $L$ , suppose we have a transduction  $f$  and we are trying to identify words  $u_1, u_2$  that do the same thing to any suffix  $v$ . The naive approach would be to check if  $f(u_1 \cdot v) = f(u_2 \cdot v)$ , but this does not work. Suppose a transduction  $f$  is such that  $f(a \cdot b) = (a, 1)(b, 2)$ ,  $f(aaa \cdot b) = (a, 1)(a, 2)(a, 3) \cdot (b, 4)$  and  $f(c \cdot b) = (c, 1)(b, 2)(b, 2)$  (we have ignored data values in this transduction). The words  $a$  and  $aaa$  do the same thing to the suffix  $b$  (the suffix is copied as it is to the output), as opposed to  $c$  (which copies the suffix twice to the output). But  $f(a \cdot b) \neq f(aaa \cdot b)$ . The problem is that we are not only comparing what  $a$  and  $aaa$  do to the suffix  $b$ , but also comparing what they do to themselves. We want to indicate in some way that we want to ignore the parts of the output that come from  $a$  or  $aaa$ :  $f(\underline{a} \mid v) = \text{left} \cdot (b, 2)$  and  $f(\underline{aaa} \mid b) = \text{left} \cdot (b, 4)$ . We have underlined  $a$  and  $aaa$  on the input side to indicate that we want to ignore them; we have replaced  $a$  and  $aaa$  in the output by **left** to indicate that they are coming from ignored parts of the input. This has been formalized as factored outputs in [5]. This is still not enough for our purpose, since the outputs  $(b, 2)$  and  $(b, 4)$  indicate that  $a$  and  $aaa$  have different lengths. This can be resolved by offsetting one of the outputs by the difference in the lengths:  $f(\underline{a} \mid v) = \text{left} \cdot (b, 2) = f_{-2}(\underline{aaa} \mid b)$ . The subscript  $-2$  in  $f_{-2}(\underline{aaa} \mid b)$  indicates that we want to offset the origins by  $-2$ . We have formalized this in the definition below, in which we have borrowed the basic definition from [5] and added data values and offsets.

► **Definition 3 (Offset factored outputs).** *Suppose  $f$  is a transduction and  $uvw$  is a data word over  $\Sigma$ . For a triple  $(\gamma, d, o)$  in  $f(uvw)$ , the abstract origin  $\text{abs}(o)$  of  $o$  is **left** (resp. **middle**, **right**) if  $o$  is in  $u$  (resp.  $v$ ,  $w$ ). The factored output  $f(\underline{u} \mid v \mid w)$  is obtained from  $f(uvw)$  by first replacing every triple  $(\gamma, d, o)$  by  $(*, *, \text{abs}(o))$  if  $\text{abs}(o) = \text{left}$  (the other triples are retained without change). Then all consecutive occurrences of  $(*, *, \text{left})$  are replaced by a single triple  $(*, *, \text{left})$  to get  $f(\underline{u} \mid v \mid w)$ . Similarly we get  $f(u \mid \underline{v} \mid w)$  and  $f(u \mid v \mid \underline{w})$  by using  $(*, *, \text{middle})$  and  $(*, *, \text{right})$  respectively. We get  $f(\underline{u} \mid v)$  and  $f(u \mid \underline{v})$  similarly, except that there is no middle part. For an integer  $z$ , we obtain  $f_z(\underline{u} \mid v)$  by replacing every triple  $(\gamma, d, o)$  by  $(\gamma, d, o + z)$  (triples  $(*, *, \text{left})$  are retained without change).*

Let  $w = (a, d_1)(a, d_2)(b, d_3)(c, d_4)$  and  $f$  be the transduction in Example 2. Then  $f(w) = (c, d_4, 4)(b, d_3, 3)(a, d_2, 2)(a, d_1, 1)$  (assuming  $d_4 \neq d_1$ ). The factored output  $f(\underline{(a, d_1)(a, d_2)} \mid (b, d_3) \mid (c, d_4))$  is  $(c, d_4, 4)(b, d_3, 3)(*, *, \text{left})$ .

It is tempting to say that two data words  $u_1$  and  $u_2$  are equivalent if for all  $v$ ,  $f(\underline{u_1} \mid v) = f_z(\underline{u_2} \mid v)$ , where  $z = |u_1| - |u_2|$ . But this does not work; continuing with the transduction  $f$  from Example 2, no two data words from the infinite set  $\{(a, d_i) \mid i \geq 1\}$  would be equivalent:  $f(\underline{(a, d_i)} \mid (a, d_i)) \neq f(\underline{(a, d_j)} \mid (a, d_i))$  for  $i \neq j$ . To get an equivalence relation with finite index, we need to realize that the important thing is not the first data value, but its (dis)equality with the last one. So we can say that for every  $i$ , there is a permutation  $\pi_i$  on data values mapping  $d_i$  to  $d_1$  such that  $f(\underline{\pi_i(a, d_i)} \mid v) = f(\underline{(a, d_1)} \mid v)$ . This will get us an equivalence relation with finite index but it is not enough, since the transducer model we build must satisfy another property: it must use only finitely many registers to remember data values. Next we examine which data values must be remembered.

Suppose  $L$  is the set of all data words in which the first and last data values are equal. Suppose a device is reading the word  $d_1 d_2 d_3 d_1$  from left to right and trying to determine whether the word belongs to  $L$  (we are ignoring letters from the finite alphabet here). The device must remember  $d_1$  when it is read first, so that it can be compared to the last data value. A machine independent characterization of what must be remembered is given in [4, Definition 2]; it says that the first occurrence of  $d_1$  in  $d_1 d_2 d_3 d_1$  is  $L$ -memorable because

replacing it with some fresh data value  $d_4$  (which doesn't occur in the word) makes a difference:  $d_1d_2d_3d_1 \in L$  but  $d_4d_2d_3d_1 \notin L$ . We adapt this concept to transductions, by suitably modifying the definition of “making a difference”.

► **Definition 4** (memorable values). *Suppose  $f$  is a transduction. A data value  $d$  is  $f$ -memorable in a data word  $u$  if there exists a data word  $v$  and a safe replacement  $d'$  for  $d$  in  $u$  such that  $f(u[d/d'] | v) \neq f(u | v)$ .*

Let  $f$  be the transduction of Example 2 and  $d_1, d_2, d_3, d'_1$  be distinct data values. We have  $f(\underline{d_1d_2d_3} | d_1) = (*, *, \text{left})(d_1, 4)$  and  $f(\underline{d'_1d_2d_3} | d_1) = (d_1, 4)(* , *, \text{left})$ . Hence,  $d_1$  is  $f$ -memorable in  $d_1d_2d_3$ .

We have to consider one more phenomenon in transductions. Consider the transduction  $f$  whose output is  $\epsilon$  for inputs of length less than five. For other inputs, the output is the third (resp. fourth) data value if the first and fifth are equal (resp. unequal). Let  $d_1, d_2, d_3, d_4, d_5, d'_1$  be distinct data values. We have  $f(d_1d_2d_3d_4 | v) = \epsilon = f(d'_1d_2d_3d_4 | v)$  if  $v = \epsilon$  and  $f(\underline{d_1d_2d_3d_4} | v) = (*, *, \text{left}) = f(\underline{d'_1d_2d_3d_4} | v)$  otherwise. Hence,  $d_1$  is not  $f$ -memorable in  $d_1d_2d_3d_4$ . However, any device implementing  $f$  must remember  $d_1$  after reading  $d_1d_2d_3d_4$ , so that it can be compared to the fifth data value. Replacing  $d_1$  by  $d'_1$  does make a difference but we cannot detect it by comparing  $f(d_1d_2d_3d_4 | v)$  and  $f(d'_1d_2d_3d_4 | v)$ . We can detect it as follows:  $f(d_1d_2d_3d_4 | \underline{d_1}) = (d_3, 3) \neq (d_4, 4) = f(d_1d_2d_3d_4 | \underline{d_5})$ . Changing the suffix from  $d_1$  to  $d_5$  influences how the prefix  $d_1d_2d_3d_4$  is transformed (in transductions, prefixes are vulnerable to the influence of suffixes). The value  $d_1$  is also contained in the prefix  $d_1d_2$ , but  $f(d_1d_2 | \underline{v}) = f(d_1d_2 | \underline{v[d_1/d_5]})$  for all  $v$ . To detect that  $d_1d_2$  is vulnerable, we first need to append  $d_3d_4$  to  $d_1d_2$  and then have a suffix in which we substitute  $d_1$  with something else. We formalize this in the definition below; it can be related to the example above by setting  $u = d_1d_2$ ,  $u' = d_3d_4$  and  $v = d_1$ .

► **Definition 5** (vulnerable values). *A data value  $d$  is  $f$ -vulnerable in a data word  $u$  if there exist data words  $u', v$  and a data value  $d'$  such that  $d$  does not occur in  $u'$ ,  $d'$  is a safe replacement for  $d$  in  $u \cdot u' \cdot v$  and  $f(u \cdot u' | \underline{v[d/d']}) \neq f(u \cdot u' | \underline{v})$ .*

Consider the transduction  $f$  defined as  $f(u) = f_1(u) \cdot f_2(u)$ ; for  $i \in [1, 2]$ ,  $f_i$  reverses its input if the  $i^{\text{th}}$  and last data values are distinct. On other inputs,  $f_i$  is the identity ( $f_1$  is the transduction given in Example 2). In the two words  $d_1d_2d_3d_1d_2d_3$  and  $d_1d_2d_3d_2d_1d_3$ ,  $d_1$  and  $d_2$  are  $f$ -memorable. For every data word  $v$ ,  $f(d_1d_2d_3d_1d_2d_3 | v) = f(d_1d_2d_3d_2d_1d_3 | v)$ , so it is tempting to say that the two words are equivalent. But after reading  $d_1d_2d_3d_1d_2d_3$ , a transducer would remember that  $d_2$  is the latest  $f$ -memorable value it has seen. After reading  $d_1d_2d_3d_2d_1d_3$ , the transducer would remember that  $d_1$  is the latest  $f$ -memorable value it has seen. Different  $f$ -memorable values play different roles and one way to distinguish which is which is to remember the order in which they occurred last. So we distinguish between  $d_1d_2d_3d_1d_2d_3$  and  $d_1d_2d_3d_2d_1d_3$ . Suppose  $d_2, d_1$  are two data values in some data word  $u$ . We say that  $d_1$  is  *fresher*  than  $d_2$  in  $u$  if the last occurrence of  $d_1$  in  $u$  is to the right of the last occurrence of  $d_2$  in  $u$ .

► **Definition 6.** *Suppose  $f$  is a transduction and  $u$  is a data word. We say that a data value  $d$  is  $f$ -influencing in  $u$  if it is either  $f$ -memorable or  $f$ -vulnerable in  $u$ . We denote by  $\text{ifl}_f(u)$  the sequence  $d_m \cdots d_1$ , where  $\{d_m, \dots, d_1\}$  is the set of all  $f$ -influencing values in  $u$  and for all  $i \in [1, m-1]$ ,  $d_i$  is fresher than  $d_{i+1}$  in  $u$ . We call  $d_i$  the  $i^{\text{th}}$   $f$ -influencing data value in  $u$ . If a data value  $d$  is both  $f$ -vulnerable and  $f$ -memorable in  $u$ , we say that  $d$  is of type **vm**. If  $d$  is  $f$ -memorable but not  $f$ -vulnerable (resp.  $f$ -vulnerable but not  $f$ -memorable) in  $u$ , we say that  $d$  is of type **m** (resp. **v**). We denote by  $\text{aifl}_f(u)$  the sequence  $(d_m, t(d_m)) \cdots (d_1, t(d_1))$ , where  $t(d_i)$  is the type of  $d_i$  for all  $i \in [1, m]$ .*

To consider two data words  $u_1$  and  $u_2$  to be equivalent, we can insist that  $\text{aifl}_f(u_1) = \text{aifl}_f(u_2)$ . But as before, this may result in some infinite set of pairwise non-equivalent data words. We will relax the condition by saying that there must be a permutation  $\pi$  on data values such that  $\text{aifl}_f(\pi(u_2)) = \text{aifl}_f(u_1)$ . This is still not enough; we have overlooked one more thing that must be considered in such an equivalence. Recall that in transductions, prefixes are vulnerable to the influence of suffixes. So if  $u_1$  is vulnerable to changing the suffix from  $v_1$  to  $v_2$ , then  $\pi(u_2)$  must also have the same vulnerability. This is covered by the third condition in the definition below.

► **Definition 7.** For a transduction  $f$ , we define the relation  $\equiv_f$  on data words as  $u_1 \equiv_f u_2$  if there exists a permutation  $\pi$  on data values satisfying the following conditions:

- $\lambda v.f_z(\pi(u_2) \mid v) = \lambda v.f(\underline{u}_1 \mid v)$ , where  $z = |u_1| - |u_2|$ ,
- $\text{aifl}_f(\pi(u_2)) = \text{aifl}_f(u_1)$  and
- for all  $u, v_1, v_2$ ,  $f(u_1 \cdot u \mid v_1) = f(u_1 \cdot u \mid v_2)$  iff  $f(\pi(u_2) \cdot u \mid v_1) = f(\pi(u_2) \cdot u \mid v_2)$ .

As in the standard lambda calculus notation,  $\lambda v.f_z(\underline{u} \mid v)$  denotes the function that maps each input  $v$  to  $f_z(\underline{u} \mid v)$ . It is routine to verify that for any data word  $u$  and permutation  $\pi$ ,  $\pi(u) \equiv_f u$ , since  $\pi$  itself satisfies all the conditions above. We denote by  $[u]_f$  the equivalence class of  $\equiv_f$  containing  $u$ .

► **Lemma 8.** If  $f$  is invariant under permutations, then  $\equiv_f$  is an equivalence relation.

Following is the main result of this extended abstract.

► **Theorem 9.** A transduction  $f$  is implemented by a SSRT iff  $f$  satisfies the following properties: 1)  $f$  is invariant under permutations, 2)  $f$  is without data peeking, 3)  $f$  has linear blow up and 4)  $\equiv_f$  has finite index.

## 4 Constructing a SSRT from a Transduction

In this section, we prove the reverse direction of Theorem 9, by showing how to construct a SSRT that implements a transduction, if it satisfies the four conditions in the theorem. SSRTs read their input from left to right. Our first task is to get SSRTs to identify influencing data values as they are read one by one. Suppose a transducer that is intended to implement a transduction  $f$  has read a data word  $u$  and has stored in its registers the data values that are  $f$ -influencing in  $u$ . Suppose the transducer reads the next symbol  $(\sigma, e)$ . To identify the data values that are  $f$ -influencing in  $u \cdot (\sigma, e)$ , will the transducer need to read the whole data word  $u \cdot (\sigma, e)$  again? The answer turns out to be no, as the following result shows. The only data values that can possibly be  $f$ -influencing in  $u \cdot (\sigma, e)$  are  $e$  and the data values that are  $f$ -influencing in  $u$ .

► **Lemma 10.** Let  $f$  be a transduction,  $u$  be a data word,  $\sigma \in \Sigma$  and  $d, e$  be distinct data values. If  $d$  is not  $f$ -memorable (resp.  $f$ -vulnerable) in  $u$ , then  $d$  is not  $f$ -memorable (resp.  $f$ -vulnerable) in  $u \cdot (\sigma, e)$ .

Next, suppose that  $d$  is  $f$ -influencing in  $u$ . How will we get the transducer to detect whether  $d$  continues to be  $f$ -influencing in  $u \cdot (\sigma, e)$ ? The following result provides a partial answer. If  $u_1 \equiv_f u_2$  and the  $i^{\text{th}}$   $f$ -influencing value in  $u_1$  continues to be  $f$ -influencing in  $u_1 \cdot (\sigma, e)$ , then the  $i^{\text{th}}$   $f$ -influencing value in  $u_2$  continues to be  $f$ -influencing in  $u_2 \cdot (\sigma, e)$ . The following result combines many such similar results into a single one.

► **Lemma 11.** Suppose  $f$  is a transduction that is invariant under permutations and without data peeking. Suppose  $u_1, u_2$  are data words such that  $u_1 \equiv_f u_2$ ,  $\text{ifl}_f(u_1) = d_1^m d_1^{m-1} \dots d_1^1$

and  $\text{ifl}_f(u_2) = d_2^m d_2^{m-1} \dots d_2^1$ . Suppose  $d_1^0 \in D$  is not  $f$ -influencing in  $u_1$ ,  $d_2^0 \in D$  is not  $f$ -influencing in  $u_2$  and  $\sigma \in \Sigma$ . For all  $i, j \in [0, m]$ , the following are true:

1.  $d_1^i$  is  $f$ -memorable (resp.  $f$ -vulnerable) in  $u_1 \cdot (\sigma, d_1^j)$  iff  $d_2^i$  is  $f$ -memorable (resp.  $f$ -vulnerable) in  $u_2 \cdot (\sigma, d_2^j)$ .
2.  $u_1 \cdot (\sigma, d_1^j) \equiv_f u_2 \cdot (\sigma, d_2^j)$ .

If  $u_1 \equiv_f u_2$ , there exists a permutation  $\pi$  such that  $\text{aifl}_f(u_1) = \text{aifl}_f(\pi(u_2))$ . Hence, all data words in the same equivalence class of  $\equiv_f$  have the same number of  $f$ -influencing values. If  $\equiv_f$  has finite index, then there is a bound (say  $I$ ) such that any data word has at most  $I$   $f$ -influencing data values. Consider a SSRT  $S_f^{\text{ifl}}$  with the set of registers  $R = \{r_1, \dots, r_I\}$ . The states are of the form  $([u]_f, ptr)$ , where  $u$  is some data word and  $ptr : [1, |\text{ifl}_f(u)|] \rightarrow R$  is a pointer function. Let  $ptr_\perp$  be the trivial function from  $\emptyset$  to  $R$ . The transitions can be designed to satisfy the following.

► **Lemma 12.** *Suppose the SSRT  $S_f^{\text{ifl}}$  starts in the configuration  $(([e]_f, ptr_\perp), val_e, 0)$  and reads some data word  $u$ . It reaches the configuration  $(([u]_f, ptr), val, |u|)$  such that  $val(ptr(i))$  is the  $i^{\text{th}}$   $f$ -influencing value in  $u$  for all  $i \in [1, |\text{ifl}_f(u)|]$ .*

The details of constructing  $S_f^{\text{ifl}}$  can be found in the full version. In short, the idea is that we can hard code rules such as “if the data value just read is the  $i^{\text{th}}$   $f$ -influencing value in  $u$ , it continues to be  $f$ -influencing in the new data word”. Lemma 11 implies that the validity of such rules depend only on the equivalence class  $[u]_f$  containing  $u$  and does not depend on  $u$  itself. So the SSRT need not remember the entire word  $u$ ; it just remembers the equivalence class  $[u]_f$  in its control state. The SSRT can check whether the new data value is the  $i^{\text{th}}$   $f$ -influencing value in  $u$ , by comparing it with the register  $ptr(i)$ .

Next we will extend the transducer to compute the output of a transduction. Suppose the transducer has read the data word  $u$  so far. The transducer doesn’t know what is the suffix that is going to come, so whatever computation it does has to cover all possibilities. The idea is to compute  $\{f(u | v) \mid v \in (\Sigma \times D)^*\}$  and store them in data word variables, so that when it has to output  $f(u)$  at the end, it can output  $f(u | \epsilon)$ . However, this set can be infinite. If  $\equiv_f$  has finite index, we can reduce it to a finite set. Recall the transduction  $f$  from Example 2 and the infinite set of data words  $\{(a, d_i) \mid i \geq 1\}$ . For any  $i \neq j$ ,  $f(\underline{(a, d_i)} \mid (a, d_i)) \neq f(\underline{(a, d_j)} \mid (a, d_i))$  for  $i \neq j$ . But for every  $i$ , there is a permutation  $\pi_i$  on data values mapping  $d_i$  to  $d_1$  so that  $f(\underline{(\pi_i(a, d_i))} \mid v) = f(\underline{(a, d_1)} \mid v)$  for any data word  $v$ . We have revealed that all data words in  $\{(a, d_i) \mid i \geq 1\}$  are equivalent by applying a permutation to each data word, so that they all have the same  $f$ -influencing data values. We formalize this idea below.

► **Definition 13.** *Let  $f$  be a transduction and  $\Pi$  be the set of all permutations on the set of data values  $D$ . An equalizing scheme for  $f$  is a function  $E : (\Sigma \times D)^* \rightarrow \Pi$  such that there exists a sequence  $\delta_1 \delta_2 \dots$  of data values satisfying the following condition: for every data word  $u$  and every  $i \in [1, |\text{ifl}_f(u)|]$ , the  $i^{\text{th}}$   $f$ -influencing data value of  $E(u)(u)$  is  $\delta_i$ .*

Note that  $E(u)(u)$  denotes the application of the permutation  $E(u)$  to the data word  $u$ . We will write  $E(u)(u)$  as  $u_q$  for short (intended to be read as “equalized  $u$ ”). Note that  $E(u)^{-1}(u_q) = u$ . Left parts that have been equalized like this will not have arbitrary influencing data values – they will be from the sequence  $\delta_1 \delta_2 \dots$ . For the transduction in Example 2, the first data value is the only influencing value in any data word. An equalizing scheme will map the first data value of all data words to  $\delta_1$ .

The relation  $\equiv_f$  identifies two prefixes when they behave similarly. We now define a relation that serves a similar purpose, but for suffixes.

► **Definition 14.** For a transduction  $f$  and equalizing scheme  $E$ , we define the relation  $\equiv_f^E$  on data words as  $v_1 \equiv_f^E v_2$  if for every data word  $u$ ,  $f(u_q | \underline{v_1}) = f(u_q | \underline{v_2})$ .

It is routine to verify that  $\equiv_f^E$  is an equivalence relation. Saying that  $v_1$  and  $v_2$  are similar suffixes if  $f(u | \underline{v_1}) = f(u | \underline{v_2})$  for all  $u$  doesn't work; this may result in infinitely many pairwise unequivalent suffixes (just like  $\equiv_f$  may have infinite index if we don't apply permutations to prefixes). So we "equalize" the prefixes so that they have the same  $f$ -influencing data values, before checking how suffixes influence them.

► **Lemma 15.** Suppose  $f$  is a transduction satisfying all the conditions of Theorem 9. If  $E$  is an equalizing scheme for  $f$ , then  $\equiv_f^E$  has finite index.

Suppose we are trying to design a SSRT to implement a transduction  $f$ , which has the property that  $\equiv_f^E$  has finite index. The SSRT can compute the set  $\{f(u_q | \underline{v}) \mid v \in (\Sigma \times D)^*\}$ , which is finite (it is enough to consider one representative  $v$  from every equivalence class of  $\equiv_f^E$ ). At the end when the SSRT has to output  $f(u)$ , it can output  $E(u)^{-1}(f(u_q | \underline{\epsilon})) = f(u)$ . The SSRT never knows what is the next suffix; at any point of time, the next suffix could be  $\epsilon$ . So the SSRT has to apply the permutation  $E(u)^{-1}$  at each step. Letting  $V$  be a finite set of representatives from every equivalence class of  $\equiv_f^E$ , the SSRT computes  $\{f(u | \underline{E(u)^{-1}(v)}) \mid v \in V\}$  at every step.

Now suppose the SSRT has computed  $\{f(u | \underline{E(u)^{-1}(v)}) \mid v \in V\}$ , stored them in data word variables and it reads the next symbol  $(\sigma, d)$ . The SSRT has to compute  $\{f(u \cdot (\sigma, d) | \underline{E(u \cdot (\sigma, d))^{-1}(v)}) \mid v \in V\}$  from whatever it had computed for  $u$ .

To explain how the above computation is done, we use some terminology. In factored outputs of the form  $f(u | \underline{v})$ ,  $f(\underline{u} | \underline{v})$ ,  $f(\underline{u} | v | \underline{w})$  or  $f(\underline{u} | \underline{v} | \underline{w})$ , a triple is said to come from  $u$  if it has origin in  $u$  or it is the triple  $(*, *, \text{left})$ . A left block in such a factored output is a maximal infix of triples, all coming from the left part  $u$ . Similarly, a non-right block is a maximal infix of triples, none coming from the right part. Middle blocks are defined similarly. For the transduction  $f$  in Example 2,  $f((a, d_1)(b, d_2)(c, d_3))$  is  $(c, d_3, 3)(b, d_2, 2)(a, d_1, 1)$ . In  $f((a, d_1)(b, d_2) | \underline{(c, d_3)})$ ,  $(b, d_2, 2)(a, d_1, 1)$  is a left block. In  $f(\underline{(a, d_1)} | (b, d_2) | \underline{(c, d_3)})$ ,  $(b, d_2, 2)$  is a middle block. In  $f(\underline{(a, d_1)} | \underline{(b, d_2)} | \underline{(c, d_3)})$ ,  $(*, *, \text{middle})(*, *, \text{left})$  is a non-right block, consisting of one middle and one left block.

The concretization of the  $i^{\text{th}}$  left block (resp. middle block) in  $f(\underline{u} | \underline{v} | \underline{w})$  is defined to be the  $i^{\text{th}}$  left block in  $f(u | \underline{vw})$  (resp. the  $i^{\text{th}}$  middle block in  $f(\underline{u} | v | \underline{w})$ ). The concretization of the  $i^{\text{th}}$  non-right block in  $f(\underline{u} | \underline{v} | \underline{w})$  is obtained by concatenating the concretizations of the left and middle blocks that occur in the  $i^{\text{th}}$  non-right block. The following is a direct consequence of the definitions.

► **Proposition 16.** The  $i^{\text{th}}$  left block of  $f(u \cdot (\sigma, d) | \underline{v})$  is the concretization of the  $i^{\text{th}}$  non-right block of  $f(\underline{u} | \underline{(\sigma, d)} | \underline{v})$ .

For the transduction  $f$  from Example 2, the first left block of  $f((a, d_1)(b, d_2) | \underline{(c, d_3)})$  is  $(b, d_2, 2)(a, d_1, 1)$ , which is the concretization of  $(*, *, \text{middle})(*, *, \text{left})$ , the first non-right block of  $f(\underline{(a, d_1)} | \underline{(b, d_2)} | \underline{(c, d_3)})$ .

From Proposition 16, we deduce that the  $i^{\text{th}}$  left block of  $f(u \cdot (\sigma, d) | \underline{E(u \cdot (\sigma, d))^{-1}(v)})$  is the concretization of the  $i^{\text{th}}$  non-right block of  $f(\underline{u} | \underline{(\sigma, d)} | \underline{E(u \cdot (\sigma, d))^{-1}(v)})$ . The concretizations come from the left blocks of  $f(u | \underline{(\sigma, d)} \cdot \underline{E(u \cdot (\sigma, d))^{-1}(v)})$  and the middle blocks of  $f(\underline{u} | \underline{(\sigma, d)} | \underline{E(u \cdot (\sigma, d))^{-1}(v)})$ . In the absence of data values, the above two statements would be as follows: The  $i^{\text{th}}$  left block of  $f(u \cdot \sigma | \underline{v})$  is the concretization of the  $i^{\text{th}}$  non-right block of  $f(\underline{u} | \underline{\sigma} | \underline{v})$ . The concretizations come from the left blocks of  $f(u | \underline{\sigma \cdot v})$  and the middle blocks of  $f(\underline{u} | \underline{\sigma} | \underline{v})$ . This technique of incrementally computing factored

outputs was introduced in [5] for SSTs. In SSTs,  $f(u \mid \underline{\sigma \cdot v})$  would have been computed as  $f(u \mid \underline{v'})$  when  $u$  was read, where  $v'$  is some word that influences prefixes in the same way as  $\sigma \cdot v$ . But in SSRTs, only  $f(u \mid \underline{E(u)^{-1}(v)})$  would have been computed for various  $v'$ ; what we need is  $f(u \mid \underline{(\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)})$ . We work around this by proving that a  $v'$  can be computed such that  $f(u \mid \underline{(\sigma, d) \cdot E(u \cdot (\sigma, d))^{-1}(v)}) = f(u \mid \underline{E(u)^{-1}(v)})$ . This needs some technical work, which can be found in the full version. The end result is summarized below.

► **Lemma 17.** *Suppose  $f$  is a transduction satisfying all the conditions in Theorem 9,  $E$  is an equalizing scheme for  $f$ ,  $u, v$  are data words and  $(\sigma, d) \in \Sigma \times D$ . There are functions  $g_1$  and  $g_2$  such that  $f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v)}) = g_1([u]_f, \mathbf{ifl}_f(u), d, v, f(u \mid \underline{E(u)^{-1}(v)}))$ , where  $v' = g_2([u]_f, \mathbf{ifl}_f(u), d, v)$ .*

The functions  $g_1$  and  $g_2$  need to be applied by the SSRT and that is possible. For  $g_2$ , it only needs  $[u]_f$  (stored in the control state),  $\mathbf{ifl}_f(u)$  (stored in the registers),  $d$  (this is the latest data value that has been read) and  $v$  (which is from a finite set and can be hardcoded). For  $g_1$ , it additionally needs  $f(u \mid \underline{E(u)^{-1}(v)})$ , which would have been stored in one of the data word variables when  $u$  was read.

Suppose  $v_1, v_2 \in V$  and  $v' = g_2([u]_f, \mathbf{ifl}_f(u), d, v_1) = g_2([u]_f, \mathbf{ifl}_f(u), d, v_2)$ . We have  $f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v_1)}) = g_1([u]_f, \mathbf{ifl}_f(u), d, v_1, f(u \mid \underline{E(u)^{-1}(v)}))$  and  $f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v_2)})$  is equal to  $g_1([u]_f, \mathbf{ifl}_f(u), d, v_2, f(u \mid \underline{E(u)^{-1}(v)}))$ . The SSRT would have stored  $f(u \mid \underline{E(u)^{-1}(v)})$  in a data word variable and now it is needed for two computations. But in SSRTs, the contents of one data word variable cannot be used in two computations, since SSRTs are copyless. This problem is solved in [5] for SSTs using a two way transducer model equivalent to SSTs. In this two way model, the suffix can be read and there is no need to perform computations for multiple suffixes. We cannot use that technique here, since there are no known two way models equivalent to SSRTs.

We solve this problem by not performing the two computations of  $g_1$  immediately. Instead, we remember the fact that there is a multiple dependency on a single data word variable. The actual computation is delayed until the SSRT reads more symbols from the input and gathers enough information about the suffix to discard all but one of the dependencies. Suppose we have delayed computing  $f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v_1)})$  due to some dependency. After reading the next symbol,  $f(u \cdot (\sigma, d) \mid \underline{E(u \cdot (\sigma, d))^{-1}(v_1)})$  itself might be needed for multiple computations. We keep track of such nested dependencies in a tree data structure called dependency tree. Dependency trees can grow unboundedly, but if  $\equiv_f^E$  has finite index, it can be shown that some parts can be discarded from time to time to keep their size bounded. We store such reduced dependency trees as part of the control states of the SSRT. The details of this construction can be found in the full version and the end result is summarized below.

**Proof sketch of reverse direction of Theorem 9.** Let  $f$  be a transduction that satisfies all the properties stated in Theorem 9. We extend the SSRT  $S_f^{\mathbf{ifl}}$ . The states will be of the form  $([u]_f, ptr, T)$  where  $[u]_f$  and  $ptr$  are as before and  $T$  is a reduced dependency tree. The SSRT will have a finite set of data word variables  $X$ . After reading any data word  $u$ , the SSRT will reach the configuration  $(([u]_f, ptr, T), val, |u|)$  that satisfies the following property. For any equivalence class  $[v]_f^E$  of  $\equiv_f^E$ , there is a leaf node  $\theta$  of  $T$  such that the path from the root of  $T$  to  $\theta$  will determine a sequence  $z$  in  $X^*$  ( $z$  is a sequence of data word variables) and  $val(z) = f(u \mid \underline{E(u)^{-1}(v)})$  ( $val(z)$  is the concatenation of the contents of data word variables according to the sequence  $z$ ). After reading the entire input  $u$ , the SSRT outputs  $f(u) = f(u \mid \underline{E(u)^{-1}(\epsilon)})$  using the leaf of  $T$  corresponding to  $[e]_f^E$ . ◀

## 5 Properties of Transductions Implemented by SSRTs

In this section, we prove the forward direction of our main result (Theorem 9). We begin by identifying data words after reading which, a SSRT reaches similar configurations

► **Definition 18.** For a SSRT  $S$ , we define a binary relation  $\equiv_S$  on data words as follows:  $u_1 \equiv_S u_2$  if they satisfy the following conditions. Suppose  $f$  is the transduction implemented by  $S$ , which reaches the configuration  $(q_1, val_1, |u_1|)$  after reading  $u_1$  and reaches  $(q_2, val_2, |u_2|)$  after reading  $u_2$ .

1.  $q_1 = q_2$ ,
2. for any two registers  $r_1, r_2$ , we have  $val_1(r_1) = val_1(r_2)$  iff  $val_2(r_1) = val_2(r_2)$ ,
3. for any register  $r$ ,  $val_1(r)$  is the  $i^{\text{th}}$   $f$ -memorable value (resp.  $f$ -vulnerable value) for some  $i$  in  $u_1$  iff  $val_2(r)$  is the  $i^{\text{th}}$   $f$ -memorable value (resp.  $f$ -vulnerable value) in  $u_2$ ,
4. for any data word variable  $x$ , we have  $val_1(x) = \epsilon$  iff  $val_2(x) = \epsilon$  and
5. for any two subsets  $X_1, X_2 \subseteq X$  and any arrangements  $\chi_1, \chi_2$  of  $X_1, X_2$  respectively,  $val_1(\chi_1) = val_1(\chi_2)$  iff  $val_2(\chi_1) = val_2(\chi_2)$ .

An arrangement of a finite set  $X_1$  is a sequence in  $X_1^*$  in which every element of  $X_1$  occurs exactly once. It is routine to verify that  $\equiv_S$  is an equivalence relation of finite index.

Suppose a SSRT  $S$  reads a data word  $u$ , reaches the configuration  $(q, val, |u|)$  and from there, continues to read a data word  $v$ . For some data word variable  $x \in X$ , if  $val(x)$  is some data word  $z$ , then none of the transitions executed while reading  $v$  will split  $z$  – it might be appended or prepended with other data words and may be moved to other variables but never split. Suppose  $X = \{x_1, \dots, x_m\}$ . The transitions executed while reading  $v$  can arrange  $val(x_1), \dots, val(x_m)$  in various ways, possibly inserting other data words (whose origin is in  $v$ , so they will be replaced by  $(*, *, \text{right})$  in  $\llbracket S \rrbracket(u \mid v)$ ) in between. Hence, any left block of  $\llbracket S \rrbracket(u \mid v)$  is  $val(\chi)$ , where  $\chi$  is some arrangement of some subset  $X' \subseteq X$ . Recall that a left block of  $\llbracket S \rrbracket(u \mid v)$  is a maximal infix that doesn't contain  $(*, *, \text{right})$ .

**Proof sketch of forward direction of Theorem 9.** Suppose a SSRT  $S$  implements a transduction  $f$ . It can be shown that  $\equiv_S$  refines  $\equiv_f$ , so  $\equiv_f$  has finite index. The most difficult part of this proof is to prove that if  $u_1 \equiv_S u_2$ , then there exists a permutation  $\pi$  such that for all data words  $u, v_1, v_2$ ,  $f(u_1 \cdot u \mid v_1) = f(u_1 \cdot u \mid v_2)$  iff  $f(\pi(u_2) \cdot u \mid v_1) = f(\pi(u_2) \cdot u \mid v_2)$ . The idea is to show that if  $f(u_1 \cdot u \mid v_1) \neq f(u_1 \cdot u \mid v_2)$ , then for some arrangements  $\chi_1, \chi_2$  of some subsets  $X_1, X_2 \subseteq X$ ,  $val_1(\chi_1) \neq val_1(\chi_2)$  ( $val_1$  (resp.  $val_2$ ) is the valuation reached by  $S$  after reading  $u_1$  (resp.  $u_2$ )). Since  $u_1 \equiv_S u_2$ , this implies that  $val_2(\chi_1) \neq val_2(\chi_2)$ , which in turn implies that  $f(\pi(u_2) \cdot u \mid v_1) \neq f(\pi(u_2) \cdot u \mid v_2)$ . ◀

## 6 Future Work

One direction to explore is whether there is a notion of minimal canonical SSRT and if a given SSRT can be reduced to an equivalent minimal one. Adding a linear order on the data domain, logical characterization of SSRTs and studying two way transducer models with data are some more interesting studies.

Using nominal automata, techniques for finite alphabets can often be elegantly carried over to infinite alphabets, as done in [19], for example. It would be interesting to see if the same can be done for streaming transducers over infinite alphabets. Using concepts from the theory of nominal automata, recent work [6] has shown that an atom extension of streaming string transducers is equivalent to a certain class of two way transducers. This model of

transducers is a restriction of SSRTs and is robust like regular languages over finite alphabets. It would also be interesting to see how can techniques in this extended abstract be simplified to work on the transducer model presented in [6].

---

## References

- 1 R. Alur and P. Černý. Expressiveness of streaming string transducers. In *FSTTCS 2010*, volume 8 of *LIPICs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.1.
- 2 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL 2011*, POPL, pages 1–12. ACM, 2011.
- 3 D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
- 4 M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Argentina*, volume 619 of *CEUR Workshop Proceedings*, 2010.
- 5 M. Bojańczyk. Transducers with origin information. In *ICALP*, volume 8573 of *LNCS*, pages 26–37, Berlin, Heidelberg, 2014. Springer.
- 6 M. Bojańczyk and R. Stefański. Single-Use Automata and Transducers for Infinite Alphabets. In *ICALP 2020*, volume 168 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 113:1–113:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.113.
- 7 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-equivalence of two-way word transducers is in PSPACE. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPICs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.FSTTCS.2018.22.
- 8 S. Cassel, F. Howar, B. Jonsson, M. Merten, and B. Steffen. A succinct canonical register automaton model. *Journal of Logical and Algebraic Methods in Programming*, 84(1):54–66, 2015. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011.
- 9 S. Cassel, B. Jonsson, F. Howar, and B. Steffen. A succinct canonical register automaton model for data domains with binary relations. In *Automated Technology for Verification and Analysis - 10th International Symposium, 2012, Proceedings*, pages 57–71, 2012. doi:10.1007/978-3-642-33386-6\_6.
- 10 Y-F Chen, O. Lengál, T. Tan, and Z. Wu. Register automata with linear arithmetic. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.
- 11 M. Chytil and V. Jákł. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, Languages and Programming, Fourth Colloquium, University of Turku, Finland, July 18-22, 1977, Proceedings*, pages 135–147. Springer Berlin Heidelberg, 1977.
- 12 A. Durand-Gasselin and P. Habermehl. Regular transformations of data words through origin information. In B. Jacobs and C. Löding, editors, *Foundations of Software Science and Computation Structures*, pages 285–300, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- 13 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPICs*, pages 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.24.

- 14 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. On computability of data word functions defined by transducers. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 217–236. Springer, 2020. doi:10.1007/978-3-030-45231-5\_12.
- 15 N. Francez and M. Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theoretical Computer Science*, 306:155–175, 2003.
- 16 F. Howar, M. Isberner, B. Steffen, O. Bauer, and B. Jonsson. Inferring semantic interfaces of data structures. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 554–571, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 17 F. Howar, B. Steffen, B. Jonsson, and S. Cassel. Inferring canonical register automata. In V. Kuncak and A. Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 251–266, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 18 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
- 19 J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szyrwelski. Learning nominal automata. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 613–625, 2017.

# Minimising Good-For-Games Automata Is NP-Complete

Sven Schewe 

University of Liverpool, UK

<https://cgi.csc.liv.ac.uk/~sven/>

sven.schewe@liverpool.ac.uk

---

## Abstract

This paper discusses the hardness of finding minimal good-for-games (GFG) Büchi, Co-Büchi, and parity automata with state based acceptance. The problem appears to sit between finding small deterministic and finding small nondeterministic automata, where minimality is NP-complete and PSPACE-complete, respectively. However, recent work of Radi and Kupferman has shown that minimising Co-Büchi automata with transition based acceptance is tractable, which suggests that the complexity of minimising GFG automata might be *cheaper* than minimising deterministic automata.

We show for the standard state based acceptance that the minimality of a GFG automaton is NP-complete for Büchi, Co-Büchi, and parity GFG automata. The proofs are a surprisingly straight forward generalisation of the proofs from deterministic Büchi automata: they use a similar reductions, and the same hard class of languages.

**2012 ACM Subject Classification** Theory of computation → Automata over infinite objects

**Keywords and phrases** Good-for-Games Automata, Automata Minimisation

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.56

**Related Version** An earlier version is available on arXiv [9], <https://arxiv.org/abs/2003.11979>.

**Funding** This work was partly supported by the EPSRC through grant EP/P020909/1.

**Acknowledgements** Many thanks to Patrick Totzke and Karoliina Lehtinen for valuable feedback and pointers to beautiful related works, as well as the constructive feedback of the reviewers.

## 1 Introduction

Good-for-games (GFG) automata form a useful class of automata that can be used to replace deterministic automata to recognise languages in several settings, like reactive synthesis [4]. As good-for-games automata sit between deterministic and general nondeterministic automata, it stands to be expected that the complexity of their minimality also sits between the minimality of deterministic automata (NP-complete [8]) and nondeterministic automata (which is PSPACE-complete like for nondeterministic finite automata [5]). It thus came as a surprise when Radi and Kupferman showed that minimising Co-Büchi automata with transition based acceptance is tractable [7].

This raises the question whether the difference is that good-for-games automata are inherently simpler to minimise, or if it is a consequence of choosing the less common transition based acceptance. We show that the answer for the more common state based acceptance is that minimising GFG automata is as hard as minimising deterministic automata.

While extending our “inclusion in NP” argument to transition based acceptance is straight forward, our hardness proof generalises the NP-hardness proof from [8], and we will close this paper with discussing why this hardness argument does not extend to automata with transition based acceptance. This leaves the complexity of minimising transition based GFG automata (except for Co-Büchi GFG automata [7]) and deterministic automata open.



© Sven Schewe;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 56; pp. 56:1–56:13



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 2 Automata

### 2.1 Nondeterministic Parity Automata

Parity automata are word automata that recognise  $\omega$ -regular languages over a finite set of symbols. A *nondeterministic parity automaton* (NPA) is a tuple  $\mathcal{P} = (\Sigma, Q, q_0, \delta, \pi)$ , where

- $\Sigma$  denotes a finite set of symbols,
- $Q$  denotes a finite set of states,
- $q_0 \in Q_+$  with  $Q_+ = Q \cup \{\perp, \top\}$  denotes a designated initial state,
- $\delta : Q_+ \times \Sigma \rightarrow 2_+^Q$  (with  $2_+^Q = 2^Q \cup \{\{\perp\}, \{\top\}\} \setminus \{\emptyset\}$ ) is a function that maps pairs of states and input letters to either a non-empty set of states, or to  $\perp$  (false, immediate rejection, blocking) or  $\top$  (true, immediate acceptance)<sup>1</sup>, such that  $\delta(\top, \sigma) = \{\top\}$  and  $\delta(\perp, \sigma) = \{\perp\}$  hold for all  $\sigma \in \Sigma$ , and
- $\pi : Q_+ \rightarrow P \subset \mathbb{N}$  is a priority function that maps states to natural numbers (mapping  $\perp$  and  $\top$  to an odd and even number, respectively), called their *priority*.

Parity automata read infinite **input words**  $\alpha = a_0a_1a_2 \dots \in \Sigma^\omega$ . (As usual,  $\omega = \mathbb{N}_0$  denotes the non-negative integers.) Their acceptance mechanism is defined in terms of runs: a run  $\rho = r_0r_1r_2 \dots \in Q_+^\omega$  of  $\mathcal{P}$  on  $\alpha$  is an  $\omega$ -word that satisfies  $r_0 = q_0$  and, for all  $i \in \omega$ ,  $r_{i+1} \in \delta(r_i, a_i)$ . A run is called *accepting* if the highest number occurring infinitely often in the infinite sequence  $\pi(r_0)\pi(r_1)\pi(r_2) \dots$  is even, and *rejecting* if it is odd. An  $\omega$ -word is *accepted* by  $\mathcal{P}$  if it has an accepting run. The set of  $\omega$ -words accepted by  $\mathcal{P}$  is called its *language*, denoted  $\mathcal{L}(\mathcal{P})$ . Two automata that recognise the same language are called *language equivalent*.

We assume without loss of generality that  $\max\{P\} \leq |Q| + 1$ . (If a priority  $p \geq 2$  does not exist, we can reduce the priority of all states whose priority is strictly greater than  $p$  by 2 without affecting acceptance.)

### 2.2 Büchi and Co-Büchi Automata

Büchi and Co-Büchi automata – abbreviated NBAs and NCAs – are NPAs where the image of the priority function  $\pi$  is contained in  $\{1, 2\}$  and  $\{2, 3\}$ , respectively. In both cases, the automaton is often denoted  $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ , where  $F \subseteq Q_+$  is called (the set of) *final states* and denotes those states with the higher priority (2 for Büchi, 3 for Co-Büchi). The remaining states  $Q_+ \setminus F$  are called *non-final* states.

### 2.3 Deterministic and Good-for-Games Automata

An automaton is called *deterministic* if the image of the transition function  $\delta$  consists only of singletons (i.e. is included in  $\{\{q\} \mid q \in Q_+\}$ ). For convenience,  $\delta$  is therefore often viewed as a function  $\bar{\delta} : Q_+ \times \Sigma \rightarrow Q_+$  (with  $\delta(q, \sigma) \mapsto \{\bar{\delta}(q, \sigma)\}$ ).

A nondeterministic automaton is called *good-for-games* (GFG) if it only relies on a limited form of nondeterminism: GFG automata can make their decision of how to resolve their nondeterministic choices on the *history* at any point of a run – rather than using the knowledge of the complete word as a nondeterministic automaton normally would – without changing their language. They can be characterised in many ways, including as automata that simulate deterministic automata.

<sup>1</sup> The question whether or not an automaton can immediately accept or reject is a matter of taste. Often, immediate rejection is covered by allowing  $\delta$  to be partial while there is no immediate acceptance. We allow both – so  $\top$  and  $\perp$  are not counted as states – but treat them as accepting and rejecting sink states, respectively, for technical convenience.

We use the following formalisation: a nondeterministic automaton  $\mathcal{P} = (\Sigma, Q, q_0, \delta, \pi)$  is good-for-games if there is function  $\nu : q_0 Q_+^* \Sigma \rightarrow Q_+$  such that, for every infinite word  $\alpha = a_0 a_1 a_2 \dots \in \Sigma^\omega$ ,  $\mathcal{P}$  has an accepting run  $\rho'$  if, and only if, it has an accepting run  $\rho = r_0 r_1 r_2 \dots \in Q_+^\omega$  with  $r_0 = q_0$  and, for all  $i \in \mathbb{N}_0$ ,  $r_{i+1} = \nu(r_0, \dots, r_i; a_i)$ .

Broadly speaking, a good-for-games automaton sits in the middle between a nondeterministic and a deterministic automaton:  $\mathcal{P}$  and  $\nu$  together define a deterministic automaton (if such a  $\nu$  exists, there is a finite state one), but as the  $\nu$  does not have to be explicitly provided,  $\mathcal{P}$  can be more succinct than a deterministic automaton.

## 2.4 Automata Transformations & Conventions

For an NPA  $\mathcal{B} = (\Sigma, Q, q_0, \delta, \pi)$  and a state  $q \in Q_+$ , we denote with  $\mathcal{B}_q = (\Sigma, Q, q, \delta, \pi)$  the automaton resulting from  $\mathcal{B}$  by changing the initial state to  $q$ .

There are two standard measures for the size of an automaton  $\mathcal{P} = (\Sigma, Q, q_0, \delta, \pi)$ : the number  $|Q|$  of its states, and the size  $\sum_{q \in Q, a \in \Sigma} |\delta(q, a)|$  of its transition table.

### 3 Main Result

We show the following theorem.

► **Theorem 1.** *The following problems are NP-complete (all 20 combinations).*

1. *Given a good-for-games / deterministic parity / Büchi / Co-Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games parity automaton with at most  $k$  states / entries in its transition table?*
2. *Given a good-for-games / deterministic Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games Büchi automaton with at most  $k$  states / entries in its transition table?*
3. *Given a good-for-games / deterministic Co-Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games Co-Büchi automaton with at most  $k$  states / entries in its transition table?*

The 20 individual questions are, of course, all very similar. Note, however, that in the ten cases where a good-for-games automaton is given, its good-for-games property is not checked; instead we simply do not require the result to be correct where the given automaton is not good-for-games. In particular, the complexity of determining GFG-ness remains an open research question (except for Büchi [1] and Co-Büchi [6] automata, where it is known to be tractable).

For inclusion in NP (Section 4), the small good-for-games automaton can be guessed, and the guess can be validated with standard simulation games (Corollary 5).

NP hardness is established in Section 5 (Theorem 15), it turns out that the known hardness proof for deterministic Büchi and Co-Büchi automata can be adjusted to good-for-games automata, providing hardness for all combinations of our main theorem.

### 4 Inclusion in NP

We start with re-visiting a standard simulation game between a verifier, who wants to establish language inclusion through simulation, and a spoiler, who wants to destroy the proof. Note that the spoiler does not try to disprove language inclusion, but merely wants to show that it cannot be established through simulation.

#### 4.1 Simulation Game

For two NPAs  $\mathcal{P}^1 = (\Sigma, Q_1, q_0^1, \delta_1, \pi_1)$  and  $\mathcal{P}^2 = (\Sigma, Q_2, q_0^2, \delta_2, \pi_2)$ , we define the “ $\mathcal{P}^2$  simulates  $\mathcal{P}^1$ ” game, where a spoiler intuitively tries to show that  $\mathcal{P}^1$  accepts a word not in the language of  $\mathcal{P}^2$ , as follows.

The game is played on  $Q_1 \times Q_2 \cup Q_1 \times \Sigma \times Q_2$  and starts in  $(q_0^1, q_0^2)$ . In a state  $(q_1, q_2) \in Q_1 \times Q_2$ , the spoiler selects a letter  $\sigma \in \Sigma$  and a  $\sigma$  successor  $q_1' \in \delta(q_1, \sigma)$  of  $q_1$  for  $\mathcal{P}^1$  and moves to  $(q_1', \sigma, q_2)$ . In a state  $(q_1', \sigma, q_2) \in Q_1 \times \Sigma \times Q_2$ , the verifier selects a  $\sigma$  successor  $q_2' \in \delta(q_2, \sigma)$  of  $q_2$  for  $\mathcal{P}^2$  and moves to  $(q_1', q_2')$ .

Verifier and spoiler will together produce a play  $(q_0^1, q_0^2)(q_1^1, a_0, q_0^2)(q_1^1, q_1^2)(q_2^1, a_1, q_1^2)(q_2^1, q_2^2)(q_3^1, a_2, q_2^2)(q_3^1, q_3^2)(q_4^1, a_3, q_3^2) \dots$ . The verifier wins if, and only if, the run  $q_0^1 q_1^1 q_2^1 q_3^1 \dots$  of  $\mathcal{P}^1$  is rejecting *or* the run  $q_0^2 q_1^2 q_2^2 q_3^2 \dots$  of  $\mathcal{P}^2$  is accepting.

Simulation games have been used to validate GFG-ness right from their introduction [4].

► **Lemma 2.** *If the verifier wins the  $\mathcal{P}^2$  simulates  $\mathcal{P}^1$  game, then she wins positionally, and checking if she wins is in NP.*

This is a standard inclusion game, and similar games have e.g. been used in [6].

**Proof.** The verifier plays a game with two disjunctive (from verifier’s perspective) parity conditions (as the complement of a parity condition is a parity condition). A parity condition is in particular a Rabin condition, and the disjunction of Rabin conditions is still a Rabin condition. Thus, if the verifier can meet her parity objective, she can do so positionally<sup>2</sup> [3]. Thus, it suffices to guess the winning strategy of the verifier, and then check (in P)<sup>3</sup> if the spoiler wins his resulting one player game with two conjunctive parity conditions. ◀

► **Lemma 3.** *Given an NPA  $\mathcal{P}^1$  and a good-for-games NPA  $\mathcal{P}^2$ , checking  $\mathcal{L}(\mathcal{P}^1) \subseteq \mathcal{L}(\mathcal{P}^2)$  is in NP.*

**Proof.** Consider the “ $\mathcal{P}^2$  simulates  $\mathcal{P}^1$ ” game played on an NPA  $\mathcal{P}^1 = (\Sigma, Q_1, q_0^1, \delta_1, \pi_1)$  and a good-for-games NPA  $\mathcal{P}^2 = (\Sigma, Q_2, q_0^2, \delta_2, \pi_2)$ .

We first show that spoiler wins this if there is a word  $\alpha \in \mathcal{L}(\mathcal{P}^1) \setminus \mathcal{L}(\mathcal{P}^2)$ : in this case, spoiler can guess such a word alongside an accepting run for  $\mathcal{P}^1$  for  $\alpha$  – note that there is no accepting run of  $\mathcal{P}^2$  for  $\alpha$ , as  $\alpha \notin \mathcal{L}(\mathcal{P}^2)$ .

We finally show that verifier wins this game if  $\mathcal{L}(\mathcal{P}^2) \supseteq \mathcal{L}(\mathcal{P}^1)$ . In this case, verifier can construct the run  $q_0^2 q_1^2 q_2^2 q_3^2 \dots$  on the word  $\alpha$  the spoiler successively produces. Moreover, as  $\mathcal{P}^2$  is good-for-games, the verifier can do this independent of the transitions the spoiler selects, basing her choices instead on her good-for-games strategy  $\nu^2$ . If  $\alpha$  is in  $\mathcal{L}(\mathcal{P}^2)$ , then  $q_0^2 q_1^2 q_2^2 q_3^2 \dots$  is accepting and verifier wins. If  $\alpha$  is not in  $\mathcal{L}(\mathcal{P}^2)$ , then  $\alpha$  is not in  $\mathcal{L}(\mathcal{P}^1) \subseteq \mathcal{L}(\mathcal{P}^2)$  either; thus  $q_0^1 q_1^1 q_2^1 q_3^1 \dots$  is rejecting and verifier wins. ◀

► **Theorem 4.** *Given an NPA  $\mathcal{P}^1$  and a good-for-games NPA  $\mathcal{P}^2$ , checking if  $\mathcal{P}^1$  is good-for-games and satisfies  $\mathcal{L}(\mathcal{P}^1) = \mathcal{L}(\mathcal{P}^2)$  is in NP.*

<sup>2</sup> A strategy is called positional if it only depends on the current state, not on the history of how one got there.

<sup>3</sup> This problem is actually in NL, as the spoiler can guess the pair of winning priorities and guess a lasso-like path with an initial part, and a repeating part that starts and ends in the same state and has the correct dominating priorities for both parity conditions. (This does not have to be a cycle, as it might be necessary to visit a state once for establishing the dominating priority for either parity condition.)

A similar game is used in [6] to establish that GFG-ness can be decided in EXPTIME. The new observation here is the inclusions in NP, assuming a GFG automaton is provided.

**Proof.** We first use the previous lemma to check  $\mathcal{L}(\mathcal{P}^1) \subseteq \mathcal{L}(\mathcal{P}^2)$  in NP. For the rest of the proof, we assume that this test has been passed, such that  $\mathcal{L}(\mathcal{P}^1) \subseteq \mathcal{L}(\mathcal{P}^2)$  was established.

We then play the same game with inverse roles, i.e. the “ $\mathcal{P}^1$  simulates  $\mathcal{P}^2$ ” game. The question if verifier wins is again in NP.

If  $\mathcal{L}(\mathcal{P}^1) \neq \mathcal{L}(\mathcal{P}^2)$  holds, then the already established  $\mathcal{L}(\mathcal{P}^1) \subseteq \mathcal{L}(\mathcal{P}^2)$  entails that there is a word  $\alpha \in \mathcal{L}(\mathcal{P}^2) \setminus \mathcal{L}(\mathcal{P}^1)$ . In this case spoiler can win by guessing such a word  $\alpha \in \mathcal{L}(\mathcal{P}^2) \setminus \mathcal{L}(\mathcal{P}^1)$  alongside an accepting run for  $\mathcal{P}^2$  for  $\alpha$  – note that there is no accepting run of  $\mathcal{P}^1$  for  $\alpha$  in this case, regardless of whether or not  $\mathcal{P}^1$  is good-for-games.

If  $\mathcal{P}^1$  is good-for-games *and*  $\mathcal{L}(\mathcal{P}^1) = \mathcal{L}(\mathcal{P}^2)$  holds, then verifier wins (because  $\mathcal{L}(\mathcal{P}^2) \subseteq \mathcal{L}(\mathcal{P}^1)$  can be verified in NP using Lemma 3).

Finally, if  $\mathcal{L}(\mathcal{P}^1) = \mathcal{L}(\mathcal{P}^2)$  holds and verifier wins, then  $\mathcal{P}^1$  is good-for-games: this is because a winning strategy – like the positional strategy that exists (Lemma 2) – for verifier in the “ $\mathcal{P}^1$  simulates  $\mathcal{P}^2$ ” game transforms a good-for-games strategy  $\nu^2$  for  $\mathcal{P}^2$  into a good-for-games strategy  $\nu^1$  for  $\mathcal{P}^1$ , and  $\mathcal{P}^1$  can simply emulate the behaviour of  $\mathcal{P}^2$  using the (positional) winning strategy from of the verifier. ◀

This provides all upper bounds of Theorem 1 when taking into account that  $k$  should be smaller than the provided automaton. (If it is not, then the answer is always “yes”, as the automaton itself can be used.)

► **Corollary 5.** *All problems from Theorem 1 can be solved in NP.*

Note that this does not result in a test whether or not a given automaton is good-for-games, it merely allows, given a good-for-games automaton, to validate that a second NPA is both: good-for-games *and* language equivalent.

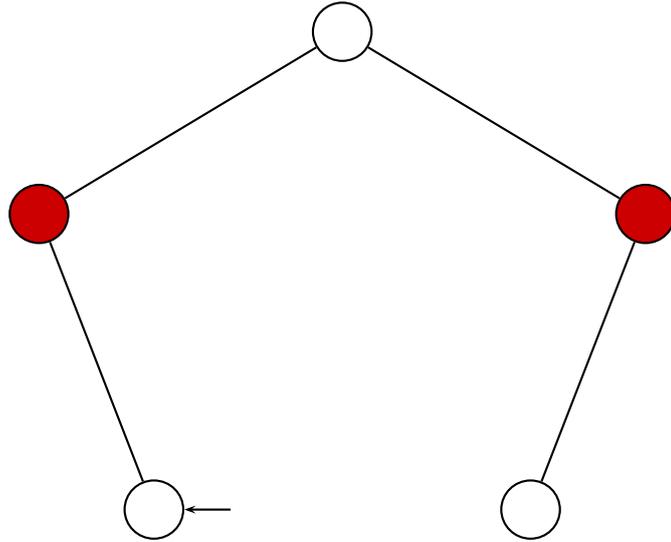
For Büchi [1] and Co-Büchi automata [6], it is tractable to check whether or not an automaton is good-for-games.

## 5 NP Hardness

In this section we generalise the hardness argument for the minimality of deterministic Büchi and Co-Büchi automata from [8]. It lifts the reduction from the problem of finding a minimal vertex cover of a graph to the minimisation of deterministic Büchi automata to a reduction to the minimisation of good-for-games automata. (A vertex cover is a set of vertices that covers at least one end point of every edge.) In the graph from Figure 1, the vertices in red and the vertices in white are both vertex covers, and the red vertices are the only minimal vertex cover. The reduction first defines the characteristic language of a simple connected graph; for technical convenience it assumes a distinguished initial vertex.

We show that the states of a good-for-games Büchi (or parity) automaton that recognises this characteristic language must satisfy side-constraints, which imply that it has at least  $2n + k$  states, where  $n$  is the number of vertices of the graph, and  $k$  is the size of its minimal vertex cover. Moreover, from a good-for-games automaton with  $s$  states, we can infer a vertex cover with size at most  $s - 2n$ .

At the same time, it is simple to construct, for a given vertex cover of size  $k$ , a deterministic Büchi automaton of size  $2n + k$  that recognises the characteristic language of this graph. (Figure 3 shows such a DBA for the example from Figure 1.) This holds in particular for the trivial vertex cover (which contains all vertices) that results in a DBA with  $3n$  states.



■ **Figure 1** A nice graph (a connected graph with a dedicated initial vertex) with a 2 vertex cover (in red). *Is a nice graph  $k$  coverable?* is an NP-complete problem.

Minimising the automaton defined by this trivial vertex cover can therefore be used to determine a minimal vertex cover for a given simple connected graph, which concludes the NP hardness argument.

Finally we show how to adjust the argument for minimal Co-Büchi automata, which – different to deterministic automata, where one can simply use the dual automaton – requires a small adjustment in the definition of the characteristic language for good-for-games automata.

Returning to the reduction known from deterministic automata, we call a non-trivial ( $|V| > 1$ ) simple undirected connected graph  $\mathcal{G}_{v_0} = (V, E)$  with a distinguished initial vertex  $v_0 \in V$  *nice*. The restriction to nice graphs leaves the problem of finding a minimal vertex cover NP-complete.

► **Lemma 6** ([8]). *The problem of checking whether a nice graph  $\mathcal{G}_{v_0}$  has a vertex cover of size  $k$  is NP-complete.*

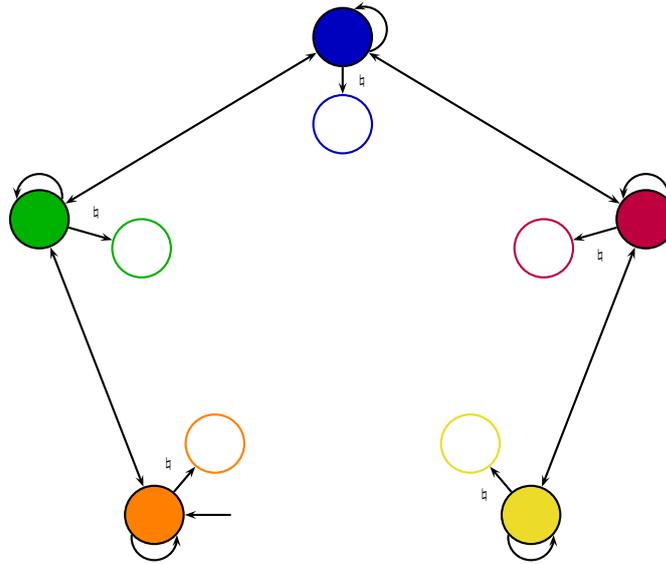
Following [8], we define the *characteristic language*  $\mathcal{L}(\mathcal{G}_{v_0})$  of a nice graph  $\mathcal{G}_{v_0}$  as the  $\omega$ -language over  $V_{\natural} = V \cup \{\natural\}$  (where  $\natural$  indicates a stop of the evaluation in the next step – it can be read “stop”) consisting of

1. all  $\omega$ -words of the form  $v_0^*v_1^+v_2^+v_3^+v_4^+\dots \in V^\omega$  with  $\{v_{i-1}, v_i\} \in E$  for all  $i \in \mathbb{N}$ , (words where  $v_0, v_1, v_2, \dots$  form an infinite path in  $\mathcal{G}_{v_0}$ ), and
2. all  $\omega$ -words that start with<sup>4</sup>  $v_0^*v_1^+v_2^+\dots v_n^+\natural v_n \in V_{\natural}^*$  with  $n \in \mathbb{N}_0$  and  $\{v_{i-1}, v_i\} \in E$  for all  $i \in \mathbb{N}$ . (Words where  $v_0, v_1, v_2, \dots, v_n$  form a finite – and potentially trivial – path in  $\mathcal{G}_{v_0}$ , followed by a  $\natural$  sign, followed by the last vertex of the path  $v_0, v_1, v_2, \dots, v_n$ , and by  $v_0$  if  $\natural$  was the first letter.)

We call the  $\omega$ -words in (1) *trace-words*, and those in (2)  $\natural$ -*words*. The trace-words are in  $V^\omega$ , while the  $\natural$ -words are in  $V_{\natural}^\omega \setminus V^\omega$ .

Figure 2 shows a deterministic Büchi automaton that recognises the  $\natural$ -words for the nice graph from Figure 1. The five colours are used as names (or: identifiers) for the vertices

<sup>4</sup> this includes words that start with  $\natural v_0$



■ **Figure 2** A deterministic Büchi automaton that recognises the  $\natural$ -words for the nice graph from Figure 1. The five colours are used as names for the vertices of the nice graph. The colour of the full (outer) vertices intuitively reflects the colour of the *previous* vertex seen while traversing an input word that can still be completed to an accepted  $\natural$  word. If the automaton reads a vertex (identified by its colour), which identifies the current vertex or a vertex adjacent to it, it updates the stored vertex to the one it has read; it blocks (moves to  $\perp$ ) when reading a different vertex. When reading  $\natural$ , it moves to the light inner vertex while keeping the stored colour/vertex, shown by the colour of its fringe. From a light (inner) vertex, it accepts (moves to  $\top$ ) if it sees the stored vertex (indicated by the colour of the fringe) next, and blocks (moves to  $\perp$ ) otherwise.

of the nice graph. The colour of the full (outer) vertices intuitively reflects the colour of the *previous* vertex seen while traversing an input word that can still be completed to an accepted  $\natural$  word (initialised to the colour of the dedicated initial vertex of the nice graph, in this case,  $\bullet$ .) If the automaton reads a vertex (here identified by its colour), which identifies either the current vertex or a vertex adjacent to it, it updates the stored vertex to the one it has read. If it reads a different vertex, which is not adjacent, it blocks (moves to  $\perp$ ).

When reading  $\natural$ , it moves to a light (inner) vertex while keeping the stored colour of the last vertex seen vertex, shown by the colour of its fringe. From a light (inner) vertex, it accepts (moves to  $\top$ ) if it sees the stored vertex (indicated by the colour of the fringe) next, and blocks (moves to  $\perp$ ) otherwise.

A word that starts with  $\bullet \bullet \natural \bullet$ , for example, is accepted, while words that start with  $\bullet \bullet \natural \bullet$  (wrong colour after  $\natural$ ) or  $\bullet \bullet \natural \bullet$  ( $\bullet$  and  $\bullet$  are not adjacent) are rejected.

Let  $\mathcal{B}$  be a parity good-for-games automaton that recognises the characteristic language of  $\mathcal{G}_{v_0} = (V, E)$ . We call a state of  $\mathcal{B}$

- a *v*-state if it can be reached upon an input word  $v_0^*v_1^+v_2^+ \dots v_n^+ \in V^*$ , with  $n \in \mathbb{N}_0$  and  $\{v_{i-1}, v_i\} \in E$  for all  $i \in \mathbb{N}$ , that ends in  $v = v_n$  (in particular, the initial state of  $\mathcal{B}$  is a  $v_0$ -state), and
- a *v* $\natural$ -state if it can be reached from a *v*-state upon reading a  $\natural$  sign.

We call the union over all *v*-states the set of *vertex-states*, and the union over all *v* $\natural$ -states the set of  $\natural$ -states.

It is not hard to define, for a given nice graph  $\mathcal{G}_{v_0} = (V, E)$  with vertex cover  $C$ , a *deterministic* Büchi automaton  $\mathcal{B}_C^{\mathcal{G}_{v_0}} = (V_{\natural}, (V \times \{n, \natural\}) \cup (C \times \{f\}), (v_0, n), \bar{\delta}, (C \times \{f\}) \cup \{\top\})$  with  $2|V| + |C|$  states that recognises the characteristic language of  $\mathcal{G}_{v_0}$  [8]. (The  $n$  and  $f$  in the state refer to non-final and final, respectively.) We simply choose

- $\bar{\delta}((v, n), v') = (v', f)$  if  $\{v, v'\} \in E$  and  $v' \in C$ ,
- $\bar{\delta}((v, n), v') = (v', n)$  if  $\{v, v'\} \in E$  and  $v' \notin C$ ,
- $\bar{\delta}((v, n), v') = (v, n)$  if  $v = v'$ ,
- $\bar{\delta}((v, n), v') = (v, \natural)$  if  $v' = \natural$ , and
- $\bar{\delta}((v, n), v') = \perp$  otherwise;
- $\bar{\delta}((v, f), v') = \bar{\delta}((v, n), v')$ , and
- $\bar{\delta}((v, \natural), v) = \top$  and  $\bar{\delta}((v, \natural), v') = \perp$  for  $v' \neq v$ .

$\mathcal{B}_C^{\mathcal{G}_{v_0}}$  simply has one  $v_{\natural}$ -state for each vertex  $v \in V$  of  $\mathcal{G}_{v_0}$ , one final  $v$ -state for each vertex in the vertex cover  $C$ , and one non-final  $v$ -state for each vertex  $v \in V$  of  $\mathcal{G}_{v_0}$ . It moves to the final (accepting) copy  $(v, f)$  for a vertex  $v \in C$  of a  $v$ -state only upon taking an edge to  $v$ , but not on a repetition of  $v$ .

Figure 3 shows a Büchi automaton that recognises the characteristic language of the nice graph from Figure 1. Different from the automaton from Figure 2, it also has to consider the trace-words, who stay in the 7 outer states (depicted as fully coloured in).

The accepting states define a cover, and a cover can be used to select final states – the automaton from Figure 3 moves to a final state whenever it “enters a vertex” from the cover shown in Figure 1. This way, every (after stuttering) infinite path sees infinitely many final states, while every (after stuttering) finite path does not. If the defining set was not a cover, then there were two adjacent states that are both not part of the cover, and the infinite path that goes back and forth between them would not be accepted.

► **Lemma 7** ([8]). *For a nice graph  $\mathcal{G}_{v_0} = (V, E)$  with initial vertex  $v_0$  and vertex cover  $C$ , the Büchi automaton  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  recognises the characteristic language of  $\mathcal{G}_{v_0}$ .*

Having seen how to get from a cover to an automaton that recognises the characteristic language of a nice graph, we now study the other direction.

► **Lemma 8.** *Let  $\mathcal{G}_{v_0} = (V, E)$  be a nice graph with initial vertex  $v_0$ , and let  $\mathcal{B} = (V_{\natural}, Q, q_0, \delta, \pi)$  be a good-for-games parity automaton that recognises the characteristic language of  $\mathcal{G}_{v_0}$ . Then the following holds:*

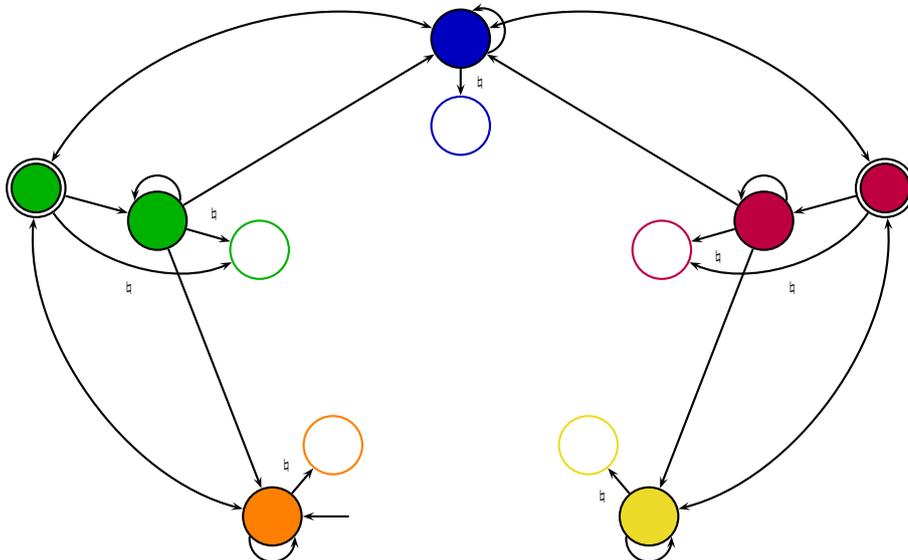
1. for all  $v$  in  $V$ , there is a  $v$ -state from which all words that start with  $\natural v$  are accepted – we call these states the core  $v$ -states;
2. for all  $v$  in  $V$ , there is a core  $v$ -state with an odd priority;
3. for all  $v \in V$  and  $w \in V_{\natural}$  with  $v \neq w$  and for every  $v$ -state  $q_v$ , words that start with  $\natural w$  are not in the language of  $\mathcal{B}_{q_v}$ ;
4. for all  $v$  in  $V$ , there is a  $v_{\natural}$ -state from which all words that start with  $v$  are accepted – we call these states the core  $v_{\natural}$ -states;
5. for all  $v$  in  $V$  and  $w \in V_{\natural}$  with  $v \neq w$  and for every  $v_{\natural}$ -state  $q_{v_{\natural}}$ , words that start with  $w$  are not in the language of  $\mathcal{B}_{q_{v_{\natural}}}$ ; and
6. for every edge  $\{v, w\} \in E$ , there is a  $v$ -state or a  $w$ -state with an even priority.

**Proof.** 1. Let  $v = v_n$  and let  $v_0, v_1, v_2, \dots, v_n$  be a path in  $\mathcal{G}_{v_0}$ . As  $\mathcal{B}$  recognises  $\mathcal{L}(\mathcal{G}_{v_0})$  and is good-for-games, it must, after having read the first  $n + 1$  or more letters of an input word  $v_0, v_1, v_2, \dots, v_n^{\omega}$  (using its good-for-games strategy  $\nu$ ), with  $\{v_i, v_{i+1}\} \in E$  for all  $i < n$ , be in a core  $v$ -state, as words that start with this and continue with  $\natural v$  are in  $\mathcal{L}(\mathcal{G}_{v_0})$ .

2. Furthermore, the run  $\mathcal{B}$  produces (using  $\nu$ ) for  $v_0, v_1, v_2, \dots, v_n^\omega$  has a dominating priority determined by its tail of core  $v$ -states, and the core  $v$ -state with the highest priority that occurs infinitely many times must have an odd priority (as the word is not in  $\mathcal{L}(\mathcal{G}_{v_0})$ ). Consequently, there must be at least one core  $v$ -state with an odd priority.
3. If (3) does not hold, a witness would provide a word accepted by  $\mathcal{B}$  but not in  $\mathcal{L}(\mathcal{G}_{v_0})$ .
4. Let  $v = v_n$  and let  $v_0, v_1, v_2, \dots, v_n$  be a path in  $\mathcal{G}_{v_0}$ . As  $\mathcal{B}$  recognises  $\mathcal{L}(\mathcal{G}_{v_0})$  and is GFG, it must, after having read the first  $n + 2$  letters of an input word that starts with  $v_0, v_1, v_2, \dots, v_n, \natural$  (using its good-for-games strategy  $\nu$ ), with  $\{v_i, v_{i+1}\} \in E$  for all  $i < n$ , be in a core  $v$ -state, as words that start with this and continue with  $v$  are in  $\mathcal{L}(\mathcal{G}_{v_0})$ .
5. If (5) does not hold, a witness would provide a word accepted by  $\mathcal{B}$  but not in  $\mathcal{L}(\mathcal{G}_{v_0})$ .
6. Let us consider an arbitrary edge  $\{v, w\} \in E$ ,  $v = v_n$ , and the run of  $\mathcal{B}$  (following  $\nu$ ) on  $v_0, v_1, v_2, \dots, v_n, (w, v)^\omega$  in  $\mathcal{L}(\mathcal{G}_{v_0})$  (i.e. for all  $i < n$ .  $\{v_i, v_{i+1}\} \in E$ ). The run must be accepting, and, as argued in (1), once the word alternates between  $v$  and  $w$ , the run alternates between core  $v$ -states and core  $w$ -states. Thus, the core  $v$ -state or the core  $w$ -state with the highest priority that occurs infinitely often must have an even priority.  $\blacktriangleleft$

The sixth claim implies that the set  $C$  of vertices with a core vertex-state with even priority is a vertex cover of  $\mathcal{G}_{v_0} = (V, E)$ . Thus,  $\mathcal{B}$  has at least  $|C|$  core vertex states with an even priority. (1–3) provide that  $\mathcal{B}$  has at least  $|V|$  vertex-states with odd priority, and it follows with (4+5) that there are  $|V|$  core  $\natural$ -states that are disjoint from the core vertex-states:

► **Corollary 9.** *For a good-for-games parity automaton  $\mathcal{B} = (V_{\natural}, Q, q_0, \delta, \pi)$  with  $s$  states that recognises the characteristic language of a nice graph  $\mathcal{G}_{v_0} = (V, E)$  with initial vertex  $v_0$ , the set  $C = \{v \in V \mid \text{there is a } v\text{-state with an even priority}\}$  is a vertex cover of  $\mathcal{G}_{v_0}$ , and  $\mathcal{B}$  has at least  $2|V| + |C|$  states ( $s \geq 2|V| + |C|$ ), such that  $|C| \leq s - 2|V|$  holds.  $\blacktriangleleft$*



■ **Figure 3** A minimal GFG (and deterministic) Büchi automaton that recognises the characteristic language of the nice graph from Figure 1. For a nice graph  $\mathcal{G}_{v_0} = (V, E)$ , a GFG parity automaton that recognises its characteristic language needs  $|V|$  states reached after reading (the first)  $\natural$  (the light, inner states),  $|V|$  states with odd priority reachable prior to reading the first  $\natural$ , and, broadly speaking, sufficiently many states with even priority, such that they identify a cover. The Büchi automaton shown here is defined by the cover that contains the red states shown in Figure 1.

## 56:10 Minimising Good-For-Games Automata Is NP-Complete

Corollary 9 and Lemma 7 immediately imply:

► **Corollary 10.** *Let  $C$  be a minimal vertex cover of a nice graph  $\mathcal{G}_{v_0} = (V, E)$ . Then  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  is a minimal deterministic Büchi automaton that recognises the characteristic language of  $\mathcal{G}_{v_0}$ , and there is no good-for-games parity automaton with less states than  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  that recognises the same language. Moreover, every minimal good-for-games automaton identifies a cover  $C'$  with  $|C'| = |C|$ . ◀*

This suffices for most cases from Theorem 1, but not for the cases where the automaton given is a Co-Büchi automaton. To also cover Co-Büchi automata, we change the characteristic language to the *adjusted language*  $\mathcal{L}'(\mathcal{G}_{v_0})$  of a nice graph  $\mathcal{G}_{v_0}$  as the  $\omega$ -language over  $V_{\natural} = V \cup \{\natural\}$  that consists of

1. all  $\omega$ -words of the form  $v_0^*v_1^+v_2^+v_3^+v_4^+\dots v_n^\omega \in V^\omega$  with  $\{v_i, v_{i+1}\} \in E$  for all  $i < n$ , (words where  $v_0, v_1, v_2, \dots, v_n$  form a finite (possibly trivial) path in  $\mathcal{G}_{v_0}$ ), and
2. all  $\omega$ -words that start with<sup>5</sup>  $v_0^*v_1^+v_2^+\dots v_n^+\natural v_n \in V_{\natural}^*$  with  $n \in \mathbb{N}_0$  and  $\{v_{i-1}, v_i\} \in E$  for all  $i \in \mathbb{N}$ . (Words where  $v_0, v_1, v_2, \dots, v_n$  form a finite – and potentially trivial – path in  $\mathcal{G}_{v_0}$ , followed by a  $\natural$  sign, followed by the last vertex of the path  $v_0, v_1, v_2, \dots, v_n$ , and by  $v_0$  if  $\natural$  was the first letter.)

► **Lemma 11.** *Let  $\mathcal{G}_{v_0} = (V, E)$  be a nice graph with initial vertex  $v_0$ , and let  $\mathcal{B} = (V_{\natural}, Q, q_0, \delta, \pi)$  be a good-for-games parity automaton that recognises the adjusted language  $\mathcal{L}'(\mathcal{G}_{v_0})$  of  $\mathcal{G}_{v_0}$ . Then the following holds:*

1. for all  $v$  in  $V$ , there is a  $v$ -state from which all words that start with  $\natural v$  are accepted – we call these states the core  $v$ -states;
2. for all  $v$  in  $V$ , there is a core  $v$ -state with an **even** priority;
3. for all  $v \in V$  and  $w \in V_{\natural}$  with  $v \neq w$  and for every  $v$ -state  $q_v$ , words that start with  $\natural w$  are not in the language of  $\mathcal{B}_{q_v}$ ;
4. for all  $v$  in  $V$ , there is a  $v_{\natural}$ -state from which all words that start with  $v$  are accepted – we call these states the core  $v_{\natural}$ -states;
5. for all  $v$  in  $V$  and  $w \in V_{\natural}$  with  $v \neq w$  and for every  $v_{\natural}$ -state  $q_{v_{\natural}}$ , words that start with  $w$  are not in the language of  $\mathcal{B}_{q_{v_{\natural}}}$ ; and
6. for every edge  $\{v, w\} \in E$ , there is a  $v$ -state or a  $w$ -state with an **odd** priority.

The changes in the proof compared to Lemma 8 are simply to replace even and odd accordingly.

With the same argument as before we get the same corollary:

► **Corollary 12.** *For a good-for-games parity automaton with  $s$  states that recognises the adjusted characteristic language of a nice graph  $\mathcal{G}_{v_0} = (V, E)$  with initial vertex  $v_0$ , the set  $C = \{v \in V \mid \text{there is a } v\text{-state with an even priority}\}$  is a vertex cover of  $\mathcal{G}_{v_0}$ , and  $\mathcal{B}$  has at least  $2|V| + |C|$  states ( $s \geq 2|V| + |C|$ ), such that  $|C| \leq s - 2|V|$  holds.*

► **Lemma 13.** *For a nice graph  $\mathcal{G}_{v_0} = (V, E)$  with initial vertex  $v_0$  and vertex cover  $C$ , the Co-Büchi automaton<sup>6</sup>  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  recognises the adjusted language of  $\mathcal{G}_{v_0}$ .*

**Proof.** We argue separately that the trace-words and  $\natural$ -words accepted by  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  are exactly the trace-words and  $\natural$ -words, respectively, in  $\mathcal{L}'(\mathcal{G}_{v_0})$ .

<sup>5</sup> this includes words that start with  $\natural v_0$

<sup>6</sup> The automaton is the same as before, but read as a Co-Büchi automaton.

For a *trace-word*  $\alpha = v_1 v_2 v_3 \dots \in V^\omega$ ,  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  has the run  $(v_0, n)(v_1, x_1)(v_2, x_2)(v_3, x_3) \dots$  (with  $x_i \in \{n, f\}$  for all  $i \in \mathbb{N}$ ) if, for all  $i \in \mathbb{N}$ , either  $v_{i-1} = v_i$  or  $\{v_{i-1}, v_i\} \in E$  holds; otherwise the automaton blocks (has a tail of  $\perp$  states) at  $i_{\min}$ -th letter, where  $i_{\min}$  is the minimal  $i$  such that  $v_{i-1} \neq v_i$  and  $\{v_{i-1}, v_i\} \notin E$ . A trace-word where the automaton blocks is rejected by  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  and not in  $\mathcal{L}'(\mathcal{G}_{v_0})$ .

We now consider those trace-words, for which  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  does not block. For these words, we call the set  $I = \{i \in \mathbb{N} \mid \{v_{i-1}, v_i\} \in E\}$  transition indices. Now  $\alpha \in \mathcal{L}'(\mathcal{G}_{v_0})$  holds if, and only if,  $I$  is finite. If  $I$  is finite, we call its maximal element  $i_{\max}$ , and set  $i_{\max}$  to 0 if  $I$  is empty. The run of  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  on  $\alpha$  is then  $(v_0, n)(v_1, x_1) \dots (v_{i_{\max}-1}, x_{i_{\max}-1})(v_{i_{\max}}, x_{i_{\max}})(v_{i_{\max}}, n)^\omega$ ; it has a tail of non-final states  $(v_{i_{\max}}, n)$ , and  $\alpha$  is therefore accepted by  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$ .

If  $I$  is infinite, we use the infinite ascending chain  $i_1 < i_2 < i_3 < \dots$  with  $I = \{i_n \mid n \in \mathbb{N}\}$ . Then, for all  $k \in \mathbb{N}$ ,  $v_{i_{k-1}} \neq v_{i_k} = v_{i_{k+1}-1} \neq v_{i_{k+1}}$  holds and  $\{v_{i_k}, v_{i_{k+1}}\} \in E$ .  $\{v_{i_k}, v_{i_{k+1}}\} \in E$  entails that the cover  $C$  must contain  $v_{i_k}$  or  $v_{i_{k+1}}$ , and it follows with  $v_{i_{k-1}} \neq v_{i_k}$  and  $v_{i_{k+1}-1} \neq v_{i_{k+1}}$  that the respective position in the run is  $(v_{i_k}, f)$  or  $(v_{i_{k+1}}, f)$  (in other words:  $x_{i_k} = f$  or  $x_{i_{k+1}} = f$ ). Thus, the run contains infinitely many final states and is rejecting.

Thus, we have shown that  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  accepts the right set of trace-words. We now continue with the simpler proof that it accepts the right set of  $\natural$ -words.

First, words starting with  $\natural v_0$  are accepted and in  $\mathcal{L}'(\mathcal{G}_{v_0})$ , while words starting with  $\natural v$  and  $v \neq v_0$  are rejected and not in  $\mathcal{L}'(\mathcal{G}_{v_0})$ .

A  $\natural$ -word that starts with  $\alpha = v_1 v_2 v_3 \dots v_n \natural w \in V^+ \natural V_{\natural}$  is in  $\mathcal{L}'(\mathcal{G}_{v_0})$  if, and only if,

1.  $v_{i-1} = v_i$  or  $\{v_{i-1}, v_i\} \in E$  holds for all  $i \leq n$ , and
2.  $v_n = w$ .

If they both hold, the (accepting) run of  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  has the form  $(v_0, n)(v_1, x_1)(v_2, x_2)(v_3, x_3) \dots (v_n, x_n)(v_n, \natural)^\omega$ .

If (1) holds but (2) does not, the (rejecting) run of  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  has the form  $(v_0, n)(v_1, x_1)(v_2, x_2)(v_3, x_3) \dots (v_n, x_n)(v_n, \natural)^\omega$ .

If (1) does not hold and  $k \leq n$  is the smallest index with  $v_{i-1} \neq v_i$  and  $\{v_{i-1}, v_i\} \notin E$ , the (rejecting) run of  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  has the form  $(v_0, n)(v_1, x_1)(v_2, x_2)(v_3, x_3) \dots (v_{k-1}, x_{k-1})^\omega$ .

As this covers all cases, we get  $\mathcal{L}(\mathcal{B}_C^{\mathcal{G}_{v_0}}) = \mathcal{L}'(\mathcal{G}_{v_0})$ .  $\blacktriangleleft$

Corollary 12 and Lemma 13 immediately imply:

► **Corollary 14.** *Let  $C$  be a minimal vertex cover of a nice graph  $\mathcal{G}_{v_0} = (V, E)$ . Then  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  is a minimal deterministic Co-Büchi automaton that recognises the adjusted characteristic language of  $\mathcal{G}_{v_0}$ , and there is no good-for-games parity automaton with less states than  $\mathcal{B}_C^{\mathcal{G}_{v_0}}$  that recognises the same language. Moreover, every minimal good-for-games automaton identifies a cover  $C'$  with  $|C'| = |C|$ .*  $\blacktriangleleft$

The Corollaries 10 and 14 provide us with the hardness result.

► **Theorem 15.** *The following problems are NP hard.*

- *Given a good-for-games / deterministic Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games Büchi automaton with at most  $k$  states / entries in its transition table (all 4 combinations)?*
- *Given a good-for-games / deterministic Co-Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games Co-Büchi automaton with at most  $k$  states / entries in its transition table (all 4 combinations)?*
- *Given a good-for-games / deterministic parity / Büchi / Co-Büchi automaton and a bound  $k$ , is there a language equivalent good-for-games parity automaton with at most  $k$  states / entries in its transition table (all 12 combinations)?*

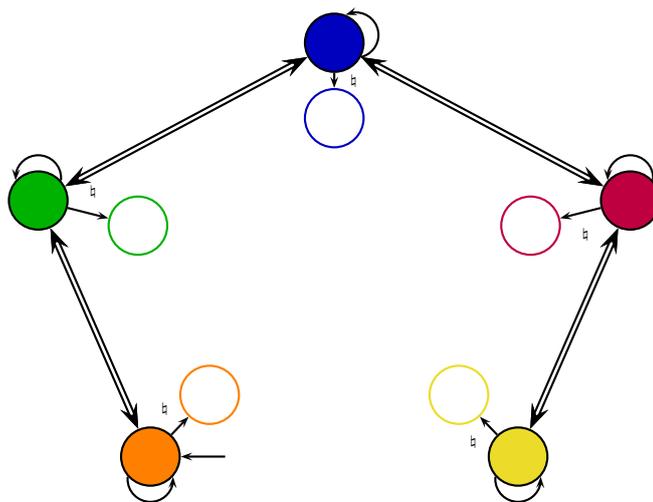
## 6 Discussion

We have established that determining if a good-for-games automaton with Büchi, Co-Büchi, or parity condition and state based acceptance is minimal, or that there is a GFG automaton with size up to  $k$ , is NP-complete. Moreover, this holds regardless of whether the starting automaton is given as a (Büchi, Co-Büchi, or parity) good-for-games automaton, or if it is presented as a (Büchi, Co-Büchi, or parity) deterministic automaton.

This drags three open questions into the limelight. The first is the complexity of testing whether or not a given nondeterministic automaton is good-for-games. Our results give no answer to this question: it simply accepts that a given automaton is good-for-games, and only guarantees a correct answer if the input is valid. GFG-ness is, however, known to be tractable for Büchi [1] and Co-Büchi [6] automata, and the extension to the more expressive class of parity good-for-games automata is active research.

It also raises the question if the difference is in good-for-games automata being inherently simpler to minimise, or if it is a property of choosing the less common transition based acceptance: the second open challenge is whether the tractability of minimising Co-Büchi good-for-games automata forebears the tractability of minimising the general class of parity good-for-games automata, while the third challenge is the question of whether NP hardness extends to transition based deterministic Büchi, Co-Büchi, and parity automata: the hard language used in this paper is not hard at all for transition based acceptance, as one can simply use final transitions between different  $v$ -states (and non-final self loops), cf. Figure 4. This could lend another argument for proliferating transition based acceptance.

In addition to the “transition vs. state based acceptance” question, another question is whether or not nondeterminism is the right starting point for GFG-ness, or if alternation is the better choice [2]. For such alternating automata, most of the succinctness and complexity questions for membership and minimisation are wide open.



■ **Figure 4** A minimal DBA with transition based acceptance for the running example.

---

**References**

---

- 1 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPICs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.FSTTCS.2018.16.
- 2 Udi Boker and Karoliina Lehtinen. Good for Games Automata: From Nondeterminism to Alternation. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 19:1–19:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPICs.CONCUR.2019.19.
- 3 E. Allen Emerson. Automata, tableaux and temporal logics. In *Proceedings of the International Conference on Logic of Programs (ICLP 1985), 17–19 June, Brooklyn, New York, USA*, volume 193 of *Lecture Notes in Computer Science*, pages 79–88. Springer-Verlag, 1985.
- 4 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. doi:10.1007/11874683\_26.
- 5 Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993. doi:10.1137/0222067.
- 6 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6\_24.
- 7 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 100:1–100:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.100.
- 8 Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPICs*, pages 400–411. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.400.
- 9 Sven Schewe. Minimising good-for-games automata is NP complete. *CoRR*, abs/2003.11979, 2020. arXiv:2003.11979.



# Static Race Detection for RTOS Applications

**Rishi Tulsyan**

Indian Institute of Science Bangalore, India  
rishitulsyan@iisc.ac.in

**Rekha Pai**

Indian Institute of Science Bangalore, India  
rekhapai@iisc.ac.in

**Deepak D'Souza<sup>1</sup>**

Indian Institute of Science Bangalore, India  
deepakd@iisc.ac.in

---

## Abstract

We present a static analysis technique for detecting data races in Real-Time Operating System (RTOS) applications. These applications are often employed in safety-critical tasks and the presence of races may lead to erroneous behaviour with serious consequences. Analyzing these applications is challenging due to the variety of non-standard synchronization mechanisms they use. We propose a technique based on the notion of an “occurs-in-between” relation between statements. This notion enables us to capture the interplay of various synchronization mechanisms. We use a pre-analysis and a small set of not-occurs-in-between patterns to detect whether two statements may race with each other. Our experimental evaluation shows that the technique is efficient and effective in identifying races with high precision.

**2012 ACM Subject Classification** Software and its engineering → Formal software verification

**Keywords and phrases** Static analysis, concurrency, data-race detection, RTOS

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.57

**Supplementary Material** [https://bitbucket.org/rishi2289/static\\_race\\_detect/](https://bitbucket.org/rishi2289/static_race_detect/)

**Acknowledgements** The second author want to thank University Grants Commission (UGC) India for the Dr. DS Kothari Post Doctoral Fellowship EN/17-18/0039.

## 1 Introduction

Real-Time Operating Systems (RTOSs) are small operating systems or microkernels that an application programmer uses as a library to create and manage the execution of multiple tasks or threads. The programs written by the application programmer are called RTOS applications and are programs typically written in C or C++ that are compiled along with the RTOS kernel library and run on bare metal processors. Much of embedded software today, ranging from home appliances to safety-critical systems like industrial automation systems and flight controller software, are implemented as such programs.

An RTOS application comprises multiple threads (even if these are typically run on a single core) and hence they need to protect against concurrency issues like data races. Two statements are involved in a data race if they are conflicting accesses to a shared memory location and can happen “simultaneously” or one after another. Data races can lead to unexpected and erroneous program behaviours, with serious consequence in safety-critical applications.

---

<sup>1</sup> corresponding author



While detecting data races is important, doing this for RTOS applications is a challenging problem. This is because these programs use a variety of non-standard synchronization mechanisms like dynamically raising and lowering priorities, suspending other tasks and the scheduler, flag-based synchronization, disabling and enabling interrupts, in addition to the more standard locks and semaphores. A look at the ArduPilot flight control software [1] which is written in C++ and runs on the ChibiOS RTOS shows several instances of *each* of these synchronization mechanisms being used. Standard techniques for race detection like lockset analysis [28] or for priority-ceiling based scheduling and flag-based synchronization [23, 22], or the disjoint-block approach of [8] for disabling interrupts, would not be precise enough as they do not handle the first two mechanisms mentioned above. Extending the disjoint-block approach for these synchronization mechanisms seems difficult.

Instead, in this work we adapt the disjoint-block approach of [8] to focus on a weaker notion of “not occurring in between”. Essentially, a statement  $s_2$  does not *occur in between* a statement  $s_1$  if it is not possible for a thread running  $s_2$  to preempt a thread while it is running  $s_1$ . If  $s_1$  and  $s_2$  cannot occur in between each other they also cannot race. We identify six patterns or rules that ensure that a statement cannot occur in between another. We take the help of a pre-analysis to identify dynamic priority ranges as well as task suspension information, for each statement in an application. Then for each pair of conflicting statements we check if the rules tell us that they cannot occur in between each other.

We have implemented our analysis for FreeRTOS applications (FreeRTOS [5] is a popular open source RTOS), and analyse several small benchmarks from the literature as well as a fragment of the ArduPilot [1] code, which we translate as a FreeRTOS application. Our analysis runs in fractions of a second with an overall precision rate of 73%.

## 2 Overview

We begin with an overview of our technique with an illustrative example adapted from a FreeRTOS demo application. The application, shown in Fig. 1, begins by creating two task threads  $t_1$  and  $t_2$  that run the task functions `prod` and `cons` respectively, both at

```

void main(...) {
1. item = count = 0;
2. xTaskCreate(prod,...,1, t1);
3. xTaskCreate(cons,...,1, t2);
4. vTaskStartScheduler();
}

void prod(...) {
10. for( ; ; ) {
11. vTaskSuspend(t2);
12. item = 5;
13. count = count+1;
14. vTaskResume(t2);
15. }
}

void cons(...) {
20. for( ; ; ) {
21. temp = item;
22. vTaskPrioritySet(NULL, 2);
23. count = count-1;
24. vTaskPrioritySet(NULL, 1);
25. }
}

```

Prio	Susp
1,1	-
1,1	-
1,1	cons
1,1	cons
1,1	cons
1,1	-
1,1	-
1,1	-
2,2	-
2,2	-
1,1	-

■ **Figure 1** A producer-consumer FreeRTOS app.

priority 1. Once the scheduler is started in line 4 of `main`, the two threads begin executing in a round-robin manner, preempting each other whenever the time slice is over (unless one thread is suspended, or the running thread has raised its priority above the other thread). The `prod` thread protects its accesses to the shared variables `item` and `count` by suspending the `cons` thread in line 11, and resuming it in line 14 after the access. Similarly, the `cons` thread protects its access to `count` by temporarily raising its priority to 2 in line 22.

We are interested in statically detecting potential data races in this application. We give a more precise definition of a race in Sec. 4, but for now we can take it to mean that two statements access a shared variable with at least one writing to it (we call these “conflicting” accesses), and these statements happen one after the other in some execution of the application.

Our analysis begins by first performing a data-flow analysis to identify the minimum and maximum dynamic priorities that each statement can run at. The computed values are shown on the second column from the right in the figure, and represent the priorities just before the statement. Thus at line 23 in `cons` the min and max priorities are both 2. We also perform a “suspended” analysis to find out at each point, which are the tasks that are guaranteed to be suspended. These values are shown in the rightmost column.

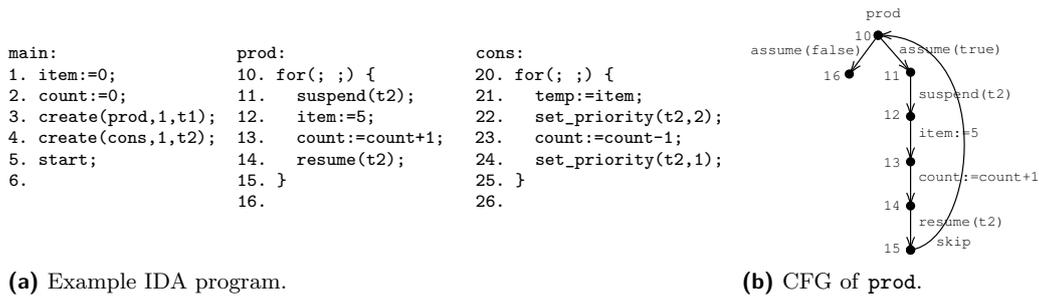
Next, for each conflicting pair of accesses  $s_1$  and  $s_2$ , we check whether  $s_2$  can “occur in between”  $s_1$ . Essentially,  $s_2$  can occur in between  $s_1$  if there is an execution in which while  $s_1$  is executing, a context-switch may happen and  $s_2$  eventually executes before the context switches back to  $s_1$ . If  $s_2$  cannot occur in between  $s_1$ , and vice-versa, then one can *rule out*  $s_1$  and  $s_2$  being involved in a race. To check the “occur in between” relation we use a small set of rules (see Fig. 4 in Sec. 5) which tell us when  $s_2$  *cannot* occur in between  $s_1$ . Thus, by the “Suspend” rule (C1), we can conclude that statements in line 21 and 23 cannot occur in between the statements in line 12 and 13 (since the `cons` task is suspended here). Similarly, by the “Priority” rule (C2), it follows that line 13 cannot occur in between line 23 (since it runs at a higher priority). This allows us to conclude that the accesses to `count` in lines 13 and 23 cannot race. However for the accesses to `item` in lines 12 and 21, we are unable to show that line 12 cannot occur in between 21, and hence our analysis declares them as potentially racy. Indeed, these two accesses are racy.

We note that analyses like [23, 8] do not handle these synchronization mechanisms and would be unable to declare the accesses in line 13 and 23 to be non-racy.

### 3 Interrupt-Driven Applications

In this section, we describe the syntax and semantics of an Interrupt-Driven Application (IDA). An IDA program is essentially a set of thread functions, which are run by dynamically created threads during execution. The functions are of two types: *task* functions which will be run by threads that are created dynamically at different priorities, and *ISR* functions which are run as Interrupt Service Routines triggered by hardware interrupts, at fixed priorities above that of task threads. There is a designated *main* function which is run by the *main* thread which is the only thread running initially. The *main* thread may create other task threads and then “start” the scheduler, at which point the created threads and ISR threads are enabled. The scheduler runs the task threads according to a highest-priority-first basis and time-slices within threads of the same priority. ISR threads can be triggered at any point of time, preempting task threads or lower priority ISR threads.

The thread functions can use a variety of commands, listed in Tab. 1, to perform computation or influence the way they are scheduled. Task threads are created using the `create` command. The command creates a new thread, which runs the specified task



■ **Figure 2** Example program and the CFG representation of `prod`.

function at the specified priority. High priority threads share execution time with low priority threads using the `set_priority`, `suspend`, and `block` commands. These commands can lead to re-scheduling of the threads, thereby giving other threads a chance to execute. The `set_priority` command sets the priority of a task thread, `suspend` suspends the execution of a task thread, and `block` (representing blocking commands like “delay” or “receive message”) blocks the execution of the current task thread. A suspended task thread can be resumed with the `resume` command. A blocked task thread resumes after a non-deterministic amount of time. Task threads can suspend and resume the scheduler with `suspend_sched` and `resume_sched`, respectively. When the scheduler is suspended the currently running task thread can be preempted only by an ISR thread, and not by other task threads. Threads can also disable and enable interrupts with `disable_int` and `enable_int`, respectively. When interrupts are disabled, no preemption can occur. Tasks can synchronize accesses to shared variables by acquiring and releasing locks with `lock` and `unlock` commands, respectively.

More formally, an IDA program  $P$  is a triple  $\langle V, M, F \rangle$  where  $V$  is a finite set of integer-valued global variables,  $M$  is a finite set of locks, and  $F$  is a finite set of thread function names, with a designated one called *main*. Each function  $A$  in  $F$  has an associated Control Flow Graph (CFG)  $G_A = (L_A, ent_A, ext_A, inst_A)$ , where  $L_A$  is the set of *locations*,  $ent_A$  and  $ext_A$  are respectively the *entry* and *exit* locations in  $L_A$ , and  $inst_A \subseteq L_A \times cmd(V, M) \times L_A$  is the set of *instructions* of the CFG. Here  $cmd(V, M)$  is the set of commands in Tab. 1 over the variables  $V$  and locks  $M$ . Each function  $A$  in  $F$  also has an associated *type*,  $type(A)$ , which is one of *task* or *ISR*. While task threads are created during execution at priorities specified in the `create` command, ISR threads run at a fixed static priority. We assume that during execution task threads can have priorities upto a constant value  $m \in \mathbb{N}$  (which we fix for all IDA programs), while ISR threads have distinct priorities which are greater than  $m$ . If  $\{f_1, \dots, f_k\}$  are the functions of type *ISR*, then without loss of generality we assume their priorities to be  $m + 1, \dots, m + k$  respectively. The IDA version of the FreeRTOS application from Fig. 1 is shown in Fig. 2.

Some notation will be useful going forward. For a program  $P$ , the instructions of  $P$ , denoted  $inst_P$ , is the union of instructions in the thread functions of  $P$ , and locations in  $P$ , denoted  $L_P$ , is the union of the locations in the thread functions of  $P$ . An IDA program allows standard integer and Boolean expressions over  $V$ . For an integer expression  $e$ , Boolean expression  $b$ , and an environment  $\phi$  for  $V$ ,  $\llbracket e \rrbracket_\phi$  denotes the integer value that  $e$  evaluates to in  $\phi$ , and  $\llbracket b \rrbracket_\phi$  denotes the Boolean value that  $b$  evaluates to in  $\phi$ . For a map  $f : X \rightarrow Y$  and elements  $x, y$  which may or may not be in  $X$  or  $Y$ , we use the notation  $f[x \mapsto y]$  to denote the function  $f' : X \cup \{x\} \rightarrow Y \cup \{y\}$  given by  $f'(x) = y$  and for all  $z$  different from  $x$ ,  $f'(z) = f(z)$ .

■ **Table 1** IDA Basic Commands.

Command	Description
<b>skip</b>	Do nothing.
$x := e$	Assign the value of expression $e$ to variable $x$ .
<b>assume</b> ( $b$ )	Enabled only if expression $b$ evaluates to <i>true</i> ; does nothing.
<b>create</b> ( $A, p, t$ )	Create task thread with func $A$ , prio $p$ , and store thread id in variable $t$ .
<b>set_priority</b> ( $t, p$ )	Set priority of task thread $t$ to $p$ . When the first parameter is NULL, set priority of current thread. Allowed only in task function.
<b>suspend</b> ( $t$ )	Suspend task thread $t$ . When the parameter is NULL, suspend current thread. Allowed only in task function.
<b>resume</b> ( $t$ )	Resume task thread $t$ . Allowed only in task function.
<b>suspendsched</b>	Suspend scheduler. Disables switching to other task threads.
<b>resumesched</b>	Resume the scheduler. Enables switching to other task threads.
<b>disableint</b>	Disable interrupts and suspend the scheduler.
<b>enableint</b>	Enable interrupts and resume the scheduler.
<b>lock</b> ( $l$ )	Acquire lock $l$ . Blocks if $l$ is not available.
<b>unlock</b> ( $l$ )	Release lock $l$ .
<b>block</b>	Block the current task thread. Re-enable after non-deterministic delay.
<b>start</b>	Start scheduler and enable interrupts. Called only by <i>main</i> .

We can now define the semantics of an IDA program  $P = \langle V, M, F \rangle$  as a labeled transition system  $\langle S, \Sigma, \Rightarrow, s_0 \rangle$ , whose components are defined as follows. Let  $f_1, \dots, f_k$  be the thread functions of type *ISR*, with priorities  $m + 1, \dots, m + k$  respectively.

The set of states  $S$  contains tuples of the form  $s = \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, pc, \phi, r, i, ss, id \rangle$ , where

- $\mathcal{B}, \mathcal{S}$ , and  $\mathcal{R}$  are sets of thread ids (which we assume to be simply integers) representing the set of *blocked* task threads, *suspended* task threads, and *ready* task threads, respectively. The sets  $\mathcal{B}, \mathcal{S}$ , and  $\mathcal{R}$  are pairwise disjoint. We denote the set of threads created so far by  $\mathcal{T} = \mathcal{B} \cup \mathcal{S} \cup \mathcal{R}$ .
- $\mathcal{P} : \mathcal{T} \rightarrow \mathbb{N}$  gives the current priority of each thread.
- $\mathcal{A} : M \rightarrow \mathcal{T}$  is a partial map giving us the thread that has acquired a particular lock.
- $\mathcal{F} : \mathcal{T} \rightarrow F$  gives the function associated with a thread.
- $pc : \mathcal{T} \rightarrow L_P$  gives the current location of a thread  $t$  in the CFG of  $\mathcal{F}(t)$ .
- $\phi \in V \rightarrow \mathbb{Z}$  is a valuation for the variables.
- $r \in \mathcal{R}$  is the currently running thread, while  $i \in \mathcal{R}$  is the interrupted task thread.
- $ss$  is a Boolean value indicating whether the scheduler is suspended ( $ss = true$ ) or not, while  $id$  is a Boolean value indicating whether interrupts are disabled ( $id = true$ ) or not.

The initial state  $s_0$  is  $\langle \emptyset, \{1, \dots, k\}, \{0\}, \{0 \mapsto 0, 1 \mapsto m + 1, \dots, k \mapsto m + k\}, \emptyset, \{0 \mapsto main, 1 \mapsto f_1, \dots, k \mapsto f_k\}, \lambda t \in \mathcal{T}.ent_{\mathcal{F}(t)}, \lambda x \in V.0, 0, 0, true, true \rangle$ . Thus initially, no threads are blocked, ISR threads  $1, \dots, k$  (with priorities  $k + 1, \dots, k + n$  respectively) are disabled, and the main thread 0 with priority 0 is ready and also running. No locks are acquired. The threads  $0, 1, \dots, k$  are associated with their functions. All the threads are at their entry locations and all variables are initialized to zero. The interrupted thread is taken to be 0, the scheduler is suspended, and interrupts are disabled.

The transition relation  $\Rightarrow$  is given as follows. Consider a state  $s$  expressed as the tuple  $s = \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, pc, \phi, r, i, ss, id \rangle$ , a thread  $t \in \mathcal{T}$ , and an instruction  $\iota = (l, c, l')$  in  $\mathcal{F}(t)$ . Then we have  $s \Rightarrow_{\iota} s'$  iff one of the following rules is satisfied. Each rule says that if the conditions on command  $c$  and state  $s$ , specified in the antecedent of a rule (above the line),

hold then  $s \Rightarrow_i s'$ , specified in the consequent of the rule (below the line), holds. We use  $task(t)$  to indicate that  $t$  is a task thread (i.e.  $(type(t) = task)$  and  $ISR(t)$  to indicate that  $t$  is an ISR thread.

In the interest of space, only few rules are shown here. The full semantics can be found in Arxiv. The ASSIGN is a simple rule on assignment statement. The ASSIGN-INT rule shows how interrupts are handled while CREATE-CS and CREATE-NS rules show how the execution of a statement can lead to context switch and no switch, respectively, and the START rule shows how the threads get running. For the ASSIGN-INT rule given below, the condition  $pc(t) = l = ent_{\mathcal{F}(t)}$  should hold while for others  $pc(t) = l$  needs to be true.

$$\frac{c = x := e \quad t = r}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, pc[t \mapsto l'], \phi[x \mapsto \llbracket e \rrbracket_\phi], r, i, ss, id \rangle} \text{ASSIGN}$$

$$\frac{c = x := e \quad t \in \mathcal{R} \quad ISR(t) \quad t \neq r \quad \mathcal{P}(t) > \mathcal{P}(r) \quad id = false}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, pc[t \mapsto l'], \phi[x \mapsto \llbracket e \rrbracket_\phi], t, r, ss, id \rangle} \text{ASSIGN-INT}$$

$$\frac{c = \text{create}(A, p, v) \quad t = r \quad task(t) \quad A \in F \quad type(A) = task \quad ts \notin \mathcal{T} \quad (p \leq \mathcal{P}(r) \vee (ss \vee id) = true)}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R} \cup \{ts\}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}[ts \mapsto A], pc[t \mapsto l', ts \mapsto ent_A], \phi[v \mapsto ts], r, i, ss, id \rangle} \text{CREATE-NS}$$

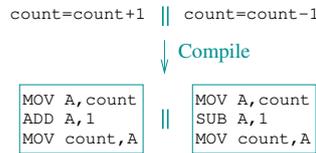
$$\frac{c = \text{create}(A, p, v) \quad t = r \quad task(t) \quad A \in F \quad type(A) = task \quad ts \notin \mathcal{T} \quad p > \mathcal{P}(r) \quad (ss \vee id) = false}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R} \cup \{ts\}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}[ts \mapsto A], pc[t \mapsto l', ts \mapsto ent_A], \phi[v \mapsto ts], ts, i, ss, id \rangle} \text{CREATE-CS}$$

$$\frac{c = \text{start} \quad t = r = 0 \quad (ss \vee id) = false \quad \exists ts \in (\mathcal{S} \cup \mathcal{R}). task(ts) \wedge \mathcal{P}(ts) = \max(\{\mathcal{P}(u) \mid u \in \mathcal{S} \cup \mathcal{R} \wedge task(u)\})}{s \Rightarrow_i \langle \mathcal{B}, \emptyset, \mathcal{S} \cup \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, pc[t \mapsto l'], \phi, ts, i, false, false \rangle} \text{START}$$

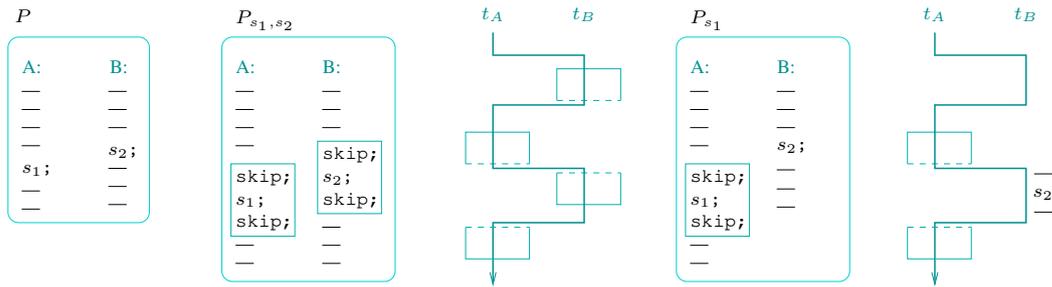
An *execution*  $\sigma$  of  $P$  is a finite sequence of transitions in the transition system defined.  $\sigma = \tau_0, \tau_1, \dots, \tau_n$ , where  $n \geq 0$  and there exists a finite sequence of states  $s_0, s_1, \dots, s_{n+1}$  in  $S$  such that  $s_0$  is the initial state and  $\tau_i = s_i \Rightarrow s_{i+1}$  for each  $0 \leq i \leq n$ .

## 4 Data Races and the Occur-in-Between Relation

We use the notion of data races introduced by Chopra et al [8], which is a general notion that applies to programs with non-standard synchronization mechanisms. The definition essentially says that two statements in a program race if (a) they are conflicting accesses to a memory location and (b) they may happen in parallel, in that notional “skip blocks” around these statements overlap with each other in some execution of the program. The definition is meant to capture the fact that when these two statements are compiled down to



instructions of a processor, the interleaving of these instructions may lead to undesirable behaviours of the program which don't correspond to any sequential execution of the two statements. For example in the figure alongside, the conflicting accesses to `count` may get compiled to the instructions shown, and the interleaving of these two blocks of instructions may lead to unexpected results like `count` getting decreased by 1 despite both blocks having completed.



■ **Figure 3** A program  $P$ ; its transformation  $P_{s_1, s_2}$ ; an execution of  $P_{s_1, s_2}$  in which the skip blocks overlap and witnesses that  $s_1$  and  $s_2$  MHP in  $P$ ; the program  $P_{s_1}$ ; and an execution of  $P_{s_1}$  which witnesses occurrence of  $s_2$  in between  $s_1$ .

We now define these notions more formally in our setting. Let us fix an IDA program  $P$ . Let  $s_1$  and  $s_2$  be two instructions in  $P$ , with associated commands  $c_1$  and  $c_2$  respectively. We restrict ourselves to the case where  $c_1$  and  $c_2$  are assignment or assume statements. We say  $s_1$  and  $s_2$  are *conflicting accesses* to a variable  $x$  if they both access  $x$  and at least one of them writes  $x$ . Let  $P_{s_1}$  denote the program obtained from  $P$  by inserting `skip` statements immediately before and after  $s_1$ . Similarly, let  $P_{s_1, s_2}$  denote the program obtained from  $P$  by inserting `skip` statements immediately before and after both  $s_1$  and  $s_2$ . We say  $s_1$  and  $s_2$  *may happen in parallel* (MHP) in  $P$  if there is an execution of  $P_{s_1, s_2}$  in which the two `skip`-blocks interleave (i.e. one block begins in between the other). These terms are illustrated in Fig. 3. We use the convention that  $A$  and  $B$  represent the static thread functions, while  $t_A$  and  $t_B$  represent dynamic threads that run the functions  $A$  and  $B$  respectively, with an optional subscript indicating the priority at which the thread was created. Finally we say  $s_1$  and  $s_2$  are involved in a *data-race* (or simply are *racy*) in  $P$ , if they are conflicting accesses that may happen in parallel in  $P$ .

It will be convenient for us to use a stronger notion than MHP called “occurs-in-between” while reasoning about IDA programs. Once again, if  $s_1$  and  $s_2$  are statements in  $P$ , we say that  $s_2$  can *occur-in-between*  $s_1$  if there is an execution of  $P_{s_1}$  in which  $s_2$  occurs sometime between the first `skip` and the second `skip` around  $s_1$ . In this case we write  $s_1 \triangleleft s_2$ , and  $s_1 \not\triangleleft s_2$  otherwise. The definition of  $s_1 \triangleleft s_2$  is illustrated in the right side of Fig. 3. While it is immediate that if  $s_2$  occurs in between  $s_1$  then they also MHP, a weaker version of the converse is also true:

► **Proposition 1.** *Let  $s_1$  and  $s_2$  be two statements in an IDA program  $P$ . Then  $s_1$  MHP  $s_2$  iff either  $s_1$  occurs in between  $s_2$  or  $s_2$  occurs in between  $s_1$ .* ◀

Thus to conclude that  $s_1$  and  $s_2$  cannot MHP (and hence not race) it is enough to show that  $s_1$  and  $s_2$  cannot occur in between each other.

## 5 Occur-In-Between Rules

In this section we focus on statically computing a conservative (i.e. under-) approximation of the *cannot-occur-in-between* relation for an IDA program, by giving rules for identifying this relation. To illustrate the typical issues we need to keep in mind while framing these rules, consider the example program alongside. Task threads  $A4$ ,  $B2$ , and  $C4$  are created at priority 4, 2, and 4 respectively. In the normal course statement  $s_2$  in  $B2$  would not be able to occur in between  $s_1$  in  $A4$  as  $A4$  runs at a higher priority than  $B2$ . However, (the thread that runs)  $C4$  may suspend  $A4$  just before it executes  $s_1$ , block itself, and allow  $B2$  to run. Thus  $s_2$  can occur in between  $s_1$ .

```

A4:   B2:   C4:
—     —     —
—     —     // t runs A4 at prio 4
s1;   s2;   suspend(t);
—     —     block;
—     —     ...
—     —     resume(t);

```

We will make use of the following terminology for an IDA program  $P$ . Let  $s$  be a statement in thread function  $A$  in  $P$ . We say  $s$  may run at priority  $p$  if there is an execution of  $P$  in which a thread  $t$  runs  $A$  and executes statement  $s$  at a priority of  $p$ . We say  $(p, q)$  is the dynamic priority of  $s$  if  $p$  and  $q$  are respectively the minimum and maximum priorities that  $s$  can run at. Similarly, we say that the dynamic priority of a thread function  $A$  (or a block of code in  $A$ ) is  $(p, q)$  if  $p$  and  $q$  are respectively the minimum and maximum priorities at which any statement in  $A$  (or the block of  $A$ ) can run. Finally, we say that a task function  $A$  may suspend another task function  $B$  in  $P$ , if  $A$  contains a statement of the form `suspend( $t$ )`, and there is an execution of  $P$  in which the statement is executed when the thread id  $t$  points to task function  $B$  (that is  $t$  runs task  $B$ ). We say the statement `suspend( $t$ )` in  $P$  must suspend (or simply *suspends*) a task  $B$  if  $t$  takes on a unique thread id at this point along any execution of  $P$ , and this thread id is the only thread that runs  $B$ . In this case, we will denote such a statement by `suspend( $B$ )`.

We now proceed to propose sufficient conditions under which one statement in an IDA program cannot occur-in-between another statement in the program. Let us fix an IDA program  $P$ . Let  $s_1$  and  $s_2$  be statements in thread functions  $A$  and  $B$  respectively ( $A$  and  $B$  could be the same thread function). The following conditions (C1)–(C6) below are meant to be sufficient conditions that ensure that  $s_2$  cannot occur in between  $s_1$ . In the rules below, by a statement  $s$  in a thread function  $A$  being enclosed in a `suspend-resume` block we mean there is a path in the CFG of  $A$  which contains  $s$ , begins with a `suspend`, ends with a `resume`, and has *no* intervening `resume` statement; and similarly for other kinds of blocks. Each of these rules is illustrated in Fig. 4.

- **C1** (Suspend Task): Each of the following conditions must hold:
  - $s_1$  is enclosed in a `suspend( $B$ )-resume( $B$ )` block with dynamic priority  $(p, q)$ ;
  - there is no task with maximum dynamic priority greater than or equal to  $p$ , that can resume  $B$ ;
  - Either no blocking statement in the `suspend( $B$ )-resume( $B$ )` block, or no other task that can resume  $B$ .
- **C2** (Priority): Each of the conditions below must hold:
  - The dynamic priorities of  $s_1$  and  $s_2$  are  $(p_1, q_1)$  and  $(p_2, q_2)$  respectively, with  $p_1 > q_2$ .
  - There is no thread body with maximum dynamic priority greater than or equal to  $p_1$  that can suspend  $A$ .
- **C3** (Flag): Each of the conditions below must hold:
  - $s_1$  is enclosed in a block  $F$  beginning with setting the variable `flag` to 1 and ending with resetting it to 0, with dynamic priority of the block being  $(p_1, q_1)$ .
  - The block  $F$  is either in the scope of a `suspendsched` command or there is no thread of priority  $\geq p_1$  that resets `flag`.
  - Either there is no blocking command before  $s_1$  in  $F$ , or no thread that can reset `flag`.
  - $s_2$  is in an `if-then` block which checks that `flag` is not set, with the block having dynamic priority  $(p_2, q_2)$ .
  - $q_1 < p_2$ .
- **C4** (Lock): Each of  $s_1$  and  $s_2$  are within a `lock( $l$ )-unlock( $l$ )` block, for some common lock  $l$ .

- **C5** (Disable Interrupts):  $s_1$  is within a `disableint-enableint` block.
- **C6** (Suspend Scheduler):  $s_1$  is within a `suspendsched-resumesched` block in a task function, and  $s_2$  is in a task function.

► **Theorem 2.** *Let  $P$  be an IDA program, and let  $s_1$  and  $s_2$  be statements in  $P$  that satisfy one of the conditions (C1) to (C6) above. Then  $s_1 \not\prec s_2$  in  $P$ .*

**Proof.** We sketch here a proof of Thm. 2 on the soundness of the conditions C1–C6. Let  $P$  be an IDA program with statements  $s_1$  and  $s_2$  satisfying one of the conditions C1–C6. We need to argue that in each case  $s_1 \not\prec s_2$ . We focus on the first three rules C1–C3 since the remaining are more standard and their soundness is easy to see.

**C1:** Suppose  $s_1$  and  $s_2$  satisfy the condition C1, and suppose there is an execution  $\rho$  of  $P$  in which  $s_2$  occurs in between  $s_1$ . Let us say  $s_1$  is executed by thread  $t_1$  and  $s_2$  by thread  $t_2$ . Then  $s_2$  must happen some time after  $t_2$  was suspended by  $t_1$ , and before  $s_1$  takes place. The only way this can happen is if:

- Some thread  $t_3$  with priority *greater than or equal to*  $p_1$  resumes  $t_2$ . But this is not possible since the condition says that there is *no* other task with dynamic priority greater than or equal to  $p_1$  which can resume  $B$ .
- $t_1$  makes a blocking call and another task runs and resumes  $t_2$ . However this is ruled out by the requirement that [there is no `block` command before  $s_1$ ] OR [there is no task other than  $A$  which can resume  $B$ ].

**C2:** Suppose  $s_1$  and  $s_2$  satisfy the condition C2, and suppose there is an execution  $\rho$  of  $P$  in which  $s_2$  occurs in between  $s_1$ . Let us say  $s_1$  is executed by thread  $t_1$  and  $s_2$  by thread  $t_2$ . Then thread  $t_2$  must preempt thread  $t_1$  during the execution of  $s_1$ . The only way this can happen is if:

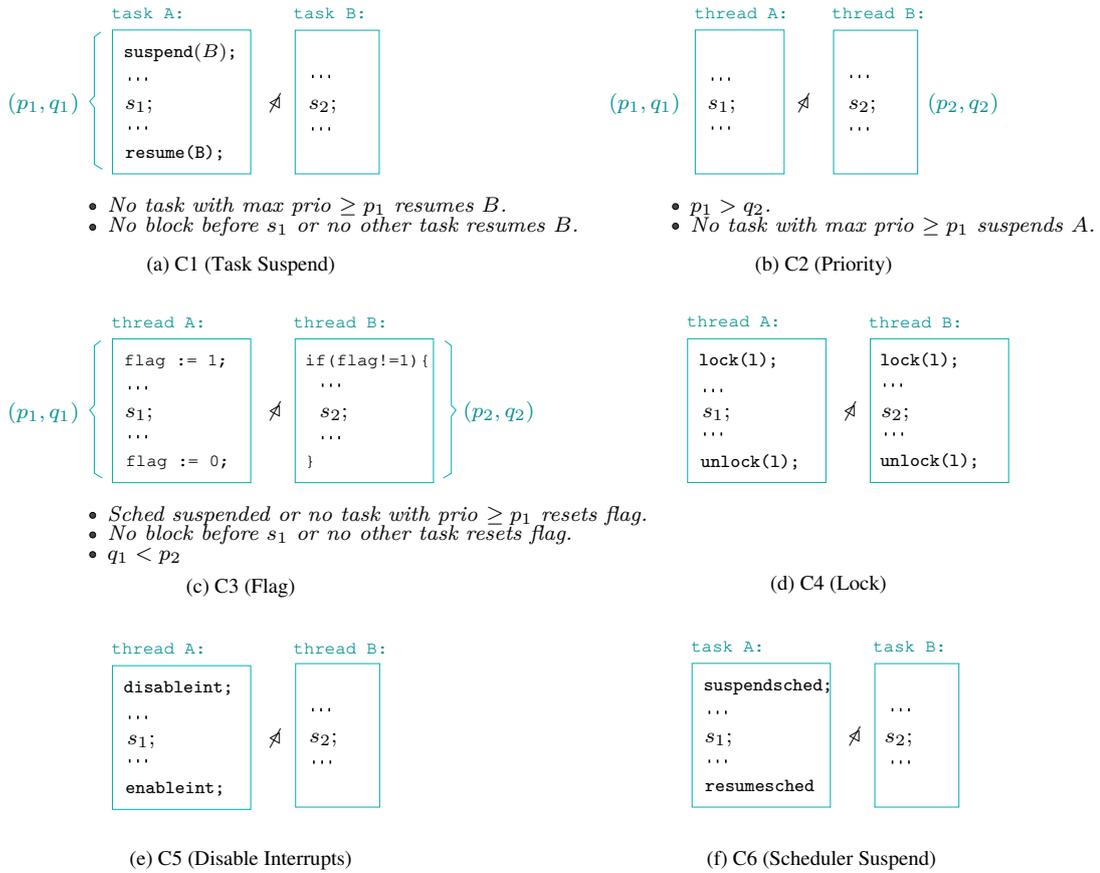
- $t_2$  with priority *greater than*  $p_1$  was blocked. It runs and preempts  $t_1$ . But this is not possible since the condition says that  $p_1 >$  maximum dynamic priority of  $t_2$ .
- $t_2$  has a priority *equal to*  $p_1$  and  $t_1$ 's time slice expires and it gets preempted by  $t_2$ . Again, this is not possible since the condition says that  $p_1 >$  maximum dynamic priority of  $t_2$ .
- Some thread  $t_3$  with priority *greater than or equal to*  $p_1$  was blocked. It runs and suspends  $t_1$ . However this is ruled out by the requirement that there is no task other than  $t_1$  with maximum dynamic priority  $\geq p_1$ , which can suspend  $t_1$ .

**C3:** Suppose  $s_1$  and  $s_2$  satisfy the condition C3, and suppose there is an execution  $\rho$  of  $P$  in which  $s_2$  occurs in between  $s_1$ . Let us say  $s_1$  is executed by thread  $t_1$  and  $s_2$  by thread  $t_2$ . Then  $s_2$  must happen some time after  $flag_1$  is set to 1 by  $t_1$ , and before  $s_1$  takes place. The only way this can happen is if:

- Some thread  $t_3$  with priority *greater than or equal to*  $p_1$  was blocked. It runs and resets  $flag_1$  to 0. But this is not possible since the condition says that  $s_1$  is either in the scope of a `suspendsched` command or there is no thread of priority  $\geq p_1$  that resets  $flag_1$ .
- $t_1$  makes a blocking call and another task runs and resets  $flag_1$  to 0. Again, this is not possible because of the requirement that [there is no `block` command before  $s_1$ ] OR [there is no task other than  $t_1$  which can reset  $flag_1$  to 0].
- Both  $t_1$  and  $t_2$  run at the same priority. Before  $t_1$  sets  $flag_1$  to 1,  $t_2$  checks  $flag_1$  and finds that it is 0, and enters the block containing  $s_2$ . Before  $t_2$  executes  $s_2$ , it's time slice expires. It gets preempted by  $t_1$  which sets  $flag_1$  to 1 and starts  $s_1$ . However this is ruled out by the requirement that  $p_2 > p_1$ .

This completes the argument. ◀

## 57:10 Data-Race Detection



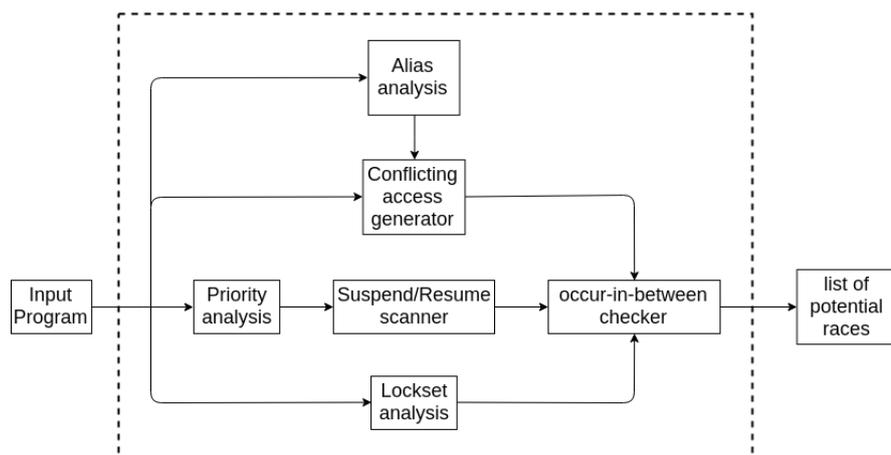
■ **Figure 4** Rules that guarantee  $s_2$  cannot occur in between  $s_1$  (i.e.  $s_1 \not\rightarrow s_2$ ).

## 6 Implementation and Evaluation

We have implemented our analysis for FreeRTOS [5] applications in a tool called RAPR (for “RTOS App Racer”). Our IDA language is closely modelled on the syntax and semantics of FreeRTOS apps, and hence we continue to use the IDA commands in place of the FreeRTOS commands in this section. Our analysis is implemented in the CIL static analysis framework [20] using OCaml, and comprises a few pre-analyses followed by the main analysis for checking the conditions. For convenience we assume that each `create` statement uses a different thread identifier.

*Priority Analysis.* The priority analysis determines the min and max dynamic priority at each statement in each thread function. This is done in 2 passes as follows. We first perform an interval-based analysis for each function  $A$ , maintaining an interval  $[p, q]$  of possible priority values at each statement. The initial interval is  $[p_0, q_0]$ , given by the min and max priorities among threads that run  $A$ . The transfer function for a `set_priority(NULL, p)` statement returns the interval  $[p, p]$ , and is identity for other statements. The join is the standard join on the interval lattice. In the second pass, for each statement `set_priority(t, p')` where  $t$  may run  $A$ , we update the interval  $[p, q]$  at each statement in  $A$  to  $[\min(p, p'), \max(q, p')]$ .

*Suspend/Resume Analysis.* This analysis determines the set of tasks which can suspend or resume a given task. We maintain a set of task functions  $\text{susplist}(A)$  and  $\text{reslist}(A)$  for each task function  $A$ , containing the set of tasks that can suspend and resume  $A$  respectively. For each task  $B$  with a `suspend(A)` or `resume(A)` statement, we add  $B$  to  $\text{susplist}(A)$  or  $\text{reslist}(A)$  respectively.



■ **Figure 5** Architecture of RAPR.

*Lockset Analysis.* A standard lockset analysis aims to compute the set of locks that are *must* held at each program point. On a `lock(l)` statement, the transfer function adds *l* to the set of locks held after this statement, while for an `unlock(l)` statement we remove *l* from the set of locks held. The join operation is simply the intersection of the locksets at the incoming points. In our setting, apart from the standard locks, we use notional locks that correspond to the different kinds of code blocks used in the rules of Fig. 4. Thus, for each `suspend(A)-resume(A)` block (rule C1) we use a notional lock  $S_A$  that we “acquire” at the `suspend(A)` statement and “release” at the `resume(A)` statement. The lockset analysis would then let us identify a `suspend(A)-resume(A)` block by the fact that the lock  $S_A$  is held throughout these statements. In a similar way we use locks  $F_{flag}^{set}$  and  $F_{flag}^{chk}$  for each flag variable *flag*, corresponding to the two blocks in rule C3; lock  $D$  for `disableint-enableint` (rule C5); and lock  $S$  for `suspendsched-resumesched` (rule C6).

*Main Analysis.* The overall analysis computes conflicting accesses by scanning for global variables having shared accesses in different threads with at least one access being a write access. We use CIL’s inbuilt alias analysis to resolve pointers to shared global variables. The information obtained from priority and lockset analysis is used to check for the cannot occur-in-between relation between the pair of conflicting accesses, using rules C1–C6. If both accesses in the pair cannot occur-in-between each other, they are declared to be non-racy; else they are declared to be potentially racy. A schematic of our implementation is shown in Fig 5.

Finally, the analysis allows a couple of command-line switches to handle some of the configuration options of FreeRTOS. Certain locks (called “mutex” locks) have a priority inheritance mechanism associated with them: when a higher priority thread is waiting on a mutex already acquired by a lower priority thread, the lower priority thread has its priority bumped up to the priority of the higher priority thread. Anticipating a need while translating nrtOSEK applications, we also allow a ceiling priority mechanism for mutexes which immediately increases the priority of the acquiring thread to the max priority of all threads that might ever acquire the mutex. To handle this we adapt the transfer function of our priority analysis for a `lock(l)` statement, when *l* is a mutex, to return  $[p, \max(q, p')]$  in the case of priority inheritance, and  $[p', p']$  in the case of ceiling priority, where  $[p, q]$  is the incoming priority interval and  $p'$  is the max priority of any task that might acquire *l*. We also provide a switch to disallow round-robin scheduling within threads of the same priority, and handle it by modifying the cannot occur-in-between conditions for C1 and C2 by replacing “>” by “≥” for the dynamic priorities.

■ **Table 2** Experimental results.

Program	LoC	Conf. acc.	True Races	RAPR				Pot. Races	Pot. Races
				Time (in s)	Pot. Races	% Elim.	% Prec.	[23]	[8]
bipedrobot.c	338	3	0	1.39	2	33.33	0.00	2	10
pe_test.c	95	4	3	0.01	3	25.00	100.00	3	7
res_test.c	110	40	8	0.03	9	77.50	88.88	9	11
tt_test.c	105	5	3	0.01	3	40.00	100.00	3	6
usb_test.c	169	0	0	0.02	0	0.00	100.00	0	52
example.c	87	13	1	0.03	1	92.30	100.00	1	61
example_fun.c	102	13	1	0.05	4	69.23	25.00	1	61
pingpong.c	112	3	0	0.01	0	100.00	100.00	0	7
counter.c	96	6	6	0.01	6	0.00	100.00	6	9
dynamic.c	429	20	2	0.13	6	70.00	33.33	16*	23
IntQueue.c	747	42	5	0.97	16	61.90	31.25	10*	24
rangefinder.c	394	16	10	0.23	10	37.50	100.00	16*	18

*Experimental Evaluation.* We tested our analysis on 12 FreeRTOS applications, shown in Tab. 2. The first 9 are nxtOSEK test programs [6] analysed in [23], which were converted to FreeRTOS programs taking care to preserve the nxtOSEK semantics which these programs use. nxtOSEK uses a priority ceiling protocol for mutex locks and no round-robin scheduling between same priority tasks. The next 2 are demo applications in FreeRTOS and finally, rangefinder.c is the converted version of an ArduPilot subsystem [1] originally in ChibiOS/C++.

The examples used by Schwarz et al. [23] consist of bipedrobot.c which is part of the control software of a self-balancing robot, pe\_test.c which tests preemptive scheduling, res\_test.c which tests resource based synchronization, tt\_test.c where tasks are time-triggered, usb\_test.c which tests usb communication, pingpong.c where two tasks set a variable to “ping” and “pong” using a mutex and counter.c where one task increments the fields of a structure and the other task resets and prints these fields. The programs example.c and example\_fun.c are examples from [23]. The FreeRTOS demo dynamic.c consists of three tasks which use different mechanisms to access a shared global counter. IntQueue.c is another FreeRTOS demo where tasks share global arrays and counters. Finally rangefinder.c is an ArduPilot subsystem with three task threads and one ISR thread which share access to a state variable and ring and bounce buffers.

Table 2 shows the results of our analysis on these programs. We ran these experiments on a Intel Core i5-8250U 1.60GHz Quad CPU machine under Ubuntu 18.04.4. Conf. acc. denotes the total number of pairs of conflicting accesses to shared global variables in the program. True races denotes the number of actual races existing in the program. RAPR “Pot. Races” denotes the number of conflicting accesses flagged to be potentially racy by the analysis. %Elim. denotes the fraction of conflicting accesses declared to be non-racy and %Prec. denotes the fraction of potential races which are actual races. Pot. Races from [23] and [8] denote the number of potential races flagged using their respective techniques.

In bipedrobot.c, the Task\_Init only runs once and hence the potential races are false positives. The decrement to digits in LowTask races with the read and write access in HighTask in pe\_test.c. In res\_test.c, the read accesses to digits are unprotected due to which it can be an actual race. The decrement of digits in LowTask is unprotected from HighTask and hence it is racy in tt\_test.c. In usb\_test.c, there are no shared accesses between tasks and hence it is trivially race-free. The races in example.c and example\_fun.c are shown

in [23]. The ping and pong tasks use a mutex to access the shared variable in `pingpong.c` and it is race-free. In `counter.c`, the fields of the global structure are accessed without any protection and hence race with each other. The first initialization of the counter by the controller task in `dynamic.c` is an actual race with the increment in the continuous increment task because both are created at the same priority and the continuous increment task can preempt the controller when it is initializing the counter. In `Intqueue.c` some accesses to the shared arrays are real races. In `rangefinder.c`, the I2C bus thread's access to the state variable is not protected from the main thread and the main thread's access to the ring buffer is not protected from the UART thread which results in a high number of actual races.

The potential races from [23] value is obtained by manually estimating the working of the idea in [23]. This is marked with a \* for the last 3 programs as their technique does not handle constructs like dynamically changing the priority of a task and hence is potentially unsound for these programs. It also results in more false positives for the `dynamic.c` and `IntQueue.c` examples as protection from synchronization mechanisms like suspending another task, disabling interrupts and suspending the scheduler is not considered. The number of potential races from [8] is obtained using their tool. The tool does not consider priorities for synchronization. Moreover it considers each task function to be run by multiple threads even if only one thread runs it in the application. These factors add to its imprecision.

In `dynamic.c` the conflicts in the continuous increment task and the limited increment task seem to be racy because they occur at the same priority but the controller task actually ensures that these two can never be in the ready state at the same time keeping one of these two suspended at all times. But this is unknown to the analysis when it encounters the conflicts as this dynamic information about the controller task cannot be made available at these points. This is the reason behind the false positives. The analysis and the test programs with the results can be found in the repository [bitbucket.org/rishi2289/static\\_race\\_detect/](https://bitbucket.org/rishi2289/static_race_detect/).

## 7 Related Work

We discuss related work grouped according to the three categories below.

*Static Race Detection.* The most closely related work is that of Schwarz *et al.* [22, 23] and Chopra *et al.* [8]. In [22, 23] Schwarz *et al.* provide a precise interprocedural data-flow analysis for checking races in OSEK-like applications that use the priority ceiling semantics. Chopra *et al.* [8] propose the notion of disjoint-blocks to detect data races and carry out data-flow analysis for FreeRTOS-like interrupt-driven *kernel APIs*. In contrast to both these works, our work handles a comprehensive variety of synchronization mechanisms, including suspend-resume and setting priorities dynamically. In addition we handle dynamic thread creation which both these works do not.

In other work in this category Chen *et al.* [7] develop a static analysis tool for race detection in binaries of interrupt-driven programs with interrupt priorities of upto two levels. The tool considers only disable-enable of interrupts as a synchronization mechanism and does not consider interleavings of task threads. Regehr and Cooper [21] describe a source-to-source translation of an interrupt-driven program to a standard multi-threaded program, and analyze the translated program for data races. However their translation is inadequate in our setting and we refer the reader to [8] for the inherent problems with such an approach. Sung *et al.* [26] propose a modular technique for static verification of interrupt-driven programs with nesting and priorities. However, the algorithm does not consider interrupt-related synchronization mechanisms nor does it consider interleavings of task threads or interaction with the ISRs. Wang *et al.* [29] present SDRacer, an automated framework that detects and

validates race conditions in interrupt-driven embedded software. The tool combines static analysis, symbolic execution, and dynamic simulation. However, it is unsound as their static analysis does not iterate to fixpoint. Mine *et al.* [18] extend Astree by employing a thread-modular static analyzer to soundly report data races in embedded C programs with mutex locks and dynamic priorities. However they do not consider interrupts and synchronization mechanisms like flag-based and suspend-resume. Finally, several papers do lockset-based static analysis for data races in classical concurrent programs [25, 11, 28, 2]. Flanagan *et al.* [12, 13] uses type system to track the lockset at each program point. However none of these techniques apply to interrupt-driven programs with non-standard synchronization mechanisms and switching semantics.

*Model-Checking.* Several researchers have used model-checking tools like Slam, Blast, and Spin to precisely model various kinds of synchronization mechanisms and detect errors exhaustively [16, 10, 15, 14, 30, 3, 19]. These technique cannot handle dynamic thread creation, and even with a small bound on the number of threads suffer from state-space explosion. Liang *et al.* [17] present an effective method to verify interrupt-driven software with nested interrupts, based on symbolic execution. The method translates a concurrent program into atomic memory read/write *events*, and then describe the interleavings of these events as a symbolic partial order expressed by a SAT/SMT formula. It is able to verify only a bounded number of interrupts.

*High-Level Race Detection.* A “high-level” race occurs when two blocks of code representing critical accesses overlap in an execution. Our definition of a data race between statements  $s_1$  and  $s_2$  in program  $P$  can thus be phrased as a high-level race on the `skip`-blocks in the augmented program  $P_{s_1, s_2}$ . Artho *et al.* [4], von Praun and Gross [27], and Pessanha *et al.* [9] study a “view”-based notion of high-level races and carry out lockset based static analysis to detect high-level races. Singh *et al.* [24] use the disjoint-block notion of [8] to detect high-level races in several RTOS *kernels*. They consider some non-standard synchronization mechanisms and also the relative scheduling priorities of specialized threads like callbacks and software interrupts. However none of these techniques handle the full gamut of synchronization mechanisms we address, and hence would be very imprecise for our applications.

## 8 Conclusions and Future Work

We have presented an efficient and precise way to detect data-races in RTOS applications that use a variety of non-standard synchronization constructs and idioms. Going forward we would like to extend our tool to be able to handle large real-life applications like ArduPilot which are written in C++ and run on the ChibiOS RTOS. We would also like to extend our technique to identify disjoint-block patterns so that we can carry out efficient data-flow analysis [8] for such applications.

---

### References

- 1 ArduPilot: Open source drone software. versatile, trusted, open. <https://ardupilot.org/>, 2020.
- 2 Martin Abadi, Cormac Flanagan, and Stephen N Freund. Types for safe locking: Static race detection for Java. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(2):207–255, 2006.
- 3 Rajeev Alur, Ken McMillan, and Doron Peled. Model-checking of correctness conditions for concurrent objects. *Information and Computation*, 160(1):167–188, 2000.
- 4 Cyrille Artho, Klaus Havelund, and Armin Biere. High-level data races. *Software Testing, Verification and Reliability*, 13(4):207–227, 2003.

- 5 Richard Barry. The FreeRTOS kernel, v10.0.0. <https://freertos.org>, 2017.
- 6 Takashi C. The NxtOSEK project. <https://sourceforge.net/projects/lejos-osek/>, 2014.
- 7 Rui Chen, Xiangying Guo, Yonghao Duan, Bin Gu, and Mengfei Yang. Static data race detection for interrupt-driven embedded software. In *Proceedings of the 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement - Companion*, SSIRI-C '11, page 47–52, USA, 2011. IEEE Computer Society.
- 8 Nikita Chopra, Rekha Pai, and Deepak D'Souza. Data races and static analysis for interrupt-driven kernels. In *Proceedings of the 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019*, volume 11423 of *Lecture Notes in Computer Science*, pages 697–723, Prague, Czech Republic, 2019. Springer. doi:10.1007/978-3-030-17184-1\_25.
- 9 Ricardo J. Dias, Vasco Pessanha, and João Lourenço. Precise detection of atomicity violations. In *Proceedings of the 8th International Haifa Verification Conference, HVC 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 8–23, Haifa, Israel, 2012. Springer. doi:10.1007/978-3-642-39611-3\_8.
- 10 Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. Precise race detection and efficient model checking using locksets. Technical Report MSR-TR-2005-118, Microsoft Research, 2005.
- 11 Dawson Engler and Ken Ashcraft. Racerx: Effective, static detection of race conditions and deadlocks. *SIGOPS Operating Systems Review*, 37(5):237–252, October 2003.
- 12 Cormac Flanagan and Stephen N. Freund. Type-based race detection for java. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2000*, pages 219–232, Vancouver, British Columbia, Canada, 2000. ACM. doi:10.1145/349299.349328.
- 13 Cormac Flanagan and Stephen N. Freund. Detecting race conditions in large programs. In *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE, 2001*, pages 90–96, Snowbird, Utah, USA, 2001. ACM. doi:10.1145/379605.379687.
- 14 Klaus Havelund, Michael R. Lowry, and John Penix. Formal analysis of a space-craft controller using SPIN. *IEEE Transactions on Software Engineering*, 27(8):749–765, 2001.
- 15 Klaus Havelund and Jens U. Skakkebak. Applying model checking in java verification. In *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, page 216–231, Berlin, Heidelberg, 1999. Springer-Verlag. doi:10.1007/3-540-48234-2\_17.
- 16 Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Race checking by context inference. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation, PLDI '04*, pages 1–13, New York, NY, USA, 2004. Association for Computing Machinery.
- 17 Lihao Liang, Tom Melham, Daniel Kroening, Peter Schrammel, and Michael Tautschnig. Effective verification for low-level software with competing interrupts. *ACM Transactions on Embedded Computing Systems*, 17(2):36:1–36:26, December 2017.
- 18 Antoine Miné, Laurent Mauborgne, Xavier Rival, Jerome Feret, Patrick Cousot, Daniel Kästner, Stephan Wilhelm, and Christian Ferdinand. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée. In *Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, 2016.
- 19 Suvam Mukherjee, Arun Kumar, and Deepak D'Souza. Detecting all high-level dataraces in an RTOS kernel. In *Proceedings of the 18th International Conference on VMCAI 2017*, volume 10145 of *Lecture Notes in Computer Science*, pages 405–423, Paris, France, 2017. Springer. doi:10.1007/978-3-319-52234-0\_22.
- 20 George Necula. CIL – infrastructure for C Program Analysis and Transformation (v. 1.3.7). <http://people.eecs.berkeley.edu/~necula/cil/>, 2002.

- 21 John Regehr and Nathan Cooperider. Interrupt verification via thread verification. *Electronic Notes in Theoretical Computer Science*, 174(9):139–150, 2007.
- 22 Martin D. Schwarz, Helmut Seidl, Vesal Vojdani, and Kalmer Apinis. Precise analysis of value-dependent synchronization in priority scheduled programs. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 21–38, San Diego, CA, USA, 2014. Springer. doi:10.1007/978-3-642-54013-4\_2.
- 23 Martin D. Schwarz, Helmut Seidl, Vesal Vojdani, Peter Lammich, and Markus Müller-Olm. Static analysis of interrupt-driven programs synchronized via the priority ceiling protocol. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 93–104, Austin, TX, USA, 2011. ACM. doi:10.1145/1926385.1926398.
- 24 Abhishek Singh, Rekha Pai, Deepak D’Souza, and Meenakshi D’Souza. Static analysis for detecting high-level races in RTOS kernels. In *Proceedings of the Formal Methods - The Next 30 Years - Third World Congress, FM 2019*, volume 11800 of *Lecture Notes in Computer Science*, pages 337–353, Porto, Portugal, 2019. Springer. doi:10.1007/978-3-030-30942-8\_21.
- 25 Nicholas Sterling. WARLOCK - A static data race analysis tool. In *Proc. Usenix Winter Technical Conference*, pages 97–106, 1993.
- 26 Chunga Sung, Markus Kusano, and Chao Wang. Modular verification of interrupt-driven software. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017*, pages 206–216, Urbana, IL, USA, 2017. IEEE Computer Society. doi:10.1109/ASE.2017.8115634.
- 27 Christoph von Praun and Thomas R. Gross. Static detection of atomicity violations in object-oriented programs. *Journal of Object Technology*, 3(6):103–122, 2004.
- 28 Jan Wen Voung, Ranjit Jhala, and Sorin Lerner. RELAY: static race detection on millions of lines of code. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007*, pages 205–214, Dubrovnik, Croatia, 2007. ACM. doi:10.1145/1287624.1287654.
- 29 Yu Wang, Linzhang Wang, Tingting Yu, Jianhua Zhao, and Xuandong Li. Automatic detection and validation of race conditions in interrupt-driven embedded software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2017*, pages 113–124, Santa Barbara, CA, USA, 2017. ACM. doi:10.1145/3092703.3092724.
- 30 Reng Zeng, Zhuo Sun, Su Liu, and Xudong He. Mepatom: A predictive analysis tool for atomicity violation using model checking. In *Proceedings of the 19th International Workshop on Model Checking Software SPIN 2012*, volume 7385 of *Lecture Notes in Computer Science*, pages 191–207, Oxford, UK, 2012. Springer. doi:10.1007/978-3-642-31759-0\_14.

## **A** Semantics

$$\begin{array}{c}
\frac{c = \text{skip}t = \text{rpc}(t) = l}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, id \rangle} \text{SKIP} \\
\frac{c = \text{skip}t \in \mathcal{R}ISR(t) \neq \text{rpc}(t) = l = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r)id = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, id \rangle} \text{SKIP-INT} \\
\frac{c = x := et = \text{rpc}(t) = l}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi[x \mapsto [e]_\phi], r, i, ss, id \rangle} \text{ASSIGN} \\
\frac{c = x := et \in \mathcal{R}ISR(t) \neq \text{rpc}(t) = l = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r)id = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi[x \mapsto [e]_\phi], t, r, ss, id \rangle} \text{ASSIGN-INT} \\
\frac{c = \text{assume}(b)t = \text{rpc}(t) = l[b]_\phi = \text{true}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, id \rangle} \text{ASSUME} \\
\frac{c = \text{assume}(b) \ t \in \mathcal{R} \ \text{ISR}(t) \ t \neq r \ \text{pc}(t) = l = \text{ent}_{\mathcal{F}(t)} \ \mathcal{P}(t) > \mathcal{P}(r) \ [b]_\phi = \text{true} \ id = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, id \rangle} \text{ASSUME-INT} \\
\frac{c = \text{create}(A, p, v) \ t = r \ \text{task}(t) \ A \in F \ \text{type}(A) = \text{task} \ ts \notin \mathcal{T} \ (p \leq \mathcal{P}(r) \vee (ss \vee id) = \text{true})}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R} \cup \{ts\}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}[ts \mapsto A], \text{pc}[t \mapsto l', ts \mapsto \text{ent}_A], \phi[v \mapsto ts], r, i, ss, id \rangle} \text{CREATE-NS} \\
\frac{c = \text{create}(A, p, v) \ t = r \ \text{task}(t) \ A \in F \ \text{type}(A) = \text{task} \ ts \notin \mathcal{T} \ p > \mathcal{P}(r) \ (ss \vee id) = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R} \cup \{ts\}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}[ts \mapsto A], \text{pc}[t \mapsto l', ts \mapsto \text{ent}_A], \phi[v \mapsto ts], ts, i, ss, id \rangle} \text{CREATE-CS} \\
\frac{c = \text{set\_priority}(ts, p) \ t = r \ \text{task}(t) \ \text{pc}(t) = l \ p \in \mathbb{N} \ \text{task}(ts) \ ts \in \mathcal{T} \ ((\mathcal{P}(r) \geq p) \vee (\mathcal{P}(r) < p \wedge ts \in (\mathcal{B} \cup \mathcal{S}))) \vee (ss \vee id) = \text{true}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, id \rangle} \text{SETP-NS} \\
\frac{c = \text{set\_priority}(ts, p) \ t = r \ \text{task}(t) \ \text{pc}(t) = l \ p \in \mathbb{N} \ \text{task}(ts) \ ts \in \mathcal{R} \ p > \mathcal{P}(r) \ (ss \vee id) = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}[ts \mapsto p], \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts, i, ss, id \rangle} \text{SETP-CS}
\end{array}$$

$$\begin{array}{c}
c = \text{suspend}(ts) \text{task}(t) = r \neq ts \text{ts} \in \mathcal{T} \text{pc}(t) = l \\
\hline
s \Rightarrow_i \langle \mathcal{B} - \{ts\}, \mathcal{S} \cup \{ts\}, \mathcal{R} - \{ts\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, id \rangle \\
\text{SUS-NS} \\
c = \text{suspend}(ts) \text{task}(t) \quad t = r = ts \quad \text{pc}(t) = l \quad (ss \vee id) = \text{false} \quad \exists ts' \in \mathcal{R}. \text{task}(ts') \wedge ts' \neq r \wedge \mathcal{P}(ts') = \max(\{\mathcal{P}(u) \mid u \in \mathcal{R} - \{r\} \wedge \text{task}(u)\}) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S} \cup \{r\}, \mathcal{R} - \{r\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts', i, ss, id \rangle \\
\text{SUS-CS} \\
c = \text{resume}(ts) \text{task}(t) \quad t = r \neq ts \quad \text{pc}(t) = l \quad ts \in (\mathcal{S} \cup \mathcal{R}) \quad ((ss \vee id) = \text{true} \vee \mathcal{P}(r) \geq \mathcal{P}(ts)) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S} - \{ts\}, \mathcal{R} \cup \{ts\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, id \rangle \\
\text{RES-NS} \\
c = \text{resume}(ts) \text{task}(t) \quad t = r \neq ts \quad \text{pc}(t) = l \quad ts \in \mathcal{S} \quad (ss \vee id) = \text{false} \quad \mathcal{P}(ts) > \mathcal{P}(r) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S} - \{ts\}, \mathcal{R} \cup \{ts\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts, i, ss, id \rangle \\
\text{RES-CS} \\
c = \text{suspendschedt} = r \text{task}(t) \text{pc}(t) = l \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, \text{true}, id \rangle \\
\text{SUSCH} \\
c = \text{resumeschedt} = r \text{task}(t) \text{pc}(t) = l \quad (\forall ts \in \mathcal{R}. \text{task}(ts) \wedge \mathcal{P}(r) \geq \mathcal{P}(ts) \vee id = \text{true}) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, \text{false}, id \rangle \\
\text{RESSCH-NS} \\
c = \text{resumesched} \quad t = r \quad \text{task}(t) \quad \text{pc}(t) = l \quad \exists ts \in \mathcal{R}. \text{task}(ts) \wedge \mathcal{P}(ts) = \max(\{\mathcal{P}(u) \mid u \in \mathcal{R} \wedge \text{task}(u)\}) \quad id = \text{false} \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts, i, \text{false}, id \rangle \\
\text{RESSCH-CS} \\
c = \text{disableint} = r \text{pc}(t) = l \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, \text{true} \rangle \\
\text{DISINT} \\
c = \text{disableint} \in \mathcal{R} \text{ISR}(t) \neq r \text{pc}(t) = l = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r) \quad id = \text{false} \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, \text{true} \rangle \\
\text{DISINT-INT} \\
c = \text{enableint} = r \text{pc}(t) = l \quad (\forall ts \in \mathcal{R}. \text{task}(ts) \wedge \mathcal{P}(r) \geq \mathcal{P}(ts) \vee ss = \text{true} \vee \text{ISR}(r)) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, \text{false} \rangle \\
\text{ENINT-NS}
\end{array}$$

$$\begin{array}{c}
\frac{c = \text{enableint } t = r \text{ task}(t) \text{ pc}(t) = l \exists ts \in \mathcal{R}. \text{task}(ts) \wedge \mathcal{P}(ts) = \max(\{\mathcal{P}(u) \mid u \in \mathcal{R} \wedge \text{task}(u)\}) \text{ ss} = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts, i, ss, \text{false} \rangle} \text{ENINT-CS} \\
\\
\frac{c = \text{enableint } t \in \mathcal{R}. \text{ISR}(t) \neq \text{rpc}(t) = l = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r) \text{id} = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, \text{false} \rangle} \text{ENINT-INT} \\
\\
\frac{c = \text{lock}(m) t = \text{rpc}(t) = l(\mathcal{A}(m) = \text{undef} \vee \mathcal{A}(m) = r)}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{A}[m \mapsto r], \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, \text{id} \rangle} \text{LOCK-AQ} \\
\\
\frac{c = \text{lock}(m) t = r \text{ task}(t) \text{ pc}(t) = l \mathcal{A}(m) = ts \neq r \text{ (ss} \vee \text{id)} = \text{false} \exists ts' \in \mathcal{R}. ts' \neq r \wedge \text{task}(ts') \wedge \mathcal{P}(ts') = \max(\{\mathcal{P}(u) \mid u \in \mathcal{R} - \{r\} \wedge \text{task}(u)\})}{s \Rightarrow_i \langle \mathcal{B} \cup \{r\}, \mathcal{S}, \mathcal{R} - \{r\}, \mathcal{A}, \mathcal{F}, \text{pc}, \phi, ts', i, ss, \text{id} \rangle} \text{LOCK-CS} \\
\\
\frac{c = \text{lock}(m) t \in \mathcal{R} \text{ ISR}(t) \text{ t} \neq r \text{ pc}(t) = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r) \mathcal{A}(m) = \text{undef} \text{id} = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}[m \mapsto t], \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, \text{id} \rangle} \text{LOCK-AQ-INT} \\
\\
\frac{c = \text{unlock}(m) t = \text{rpc}(t) = l(\mathcal{A}(m) = r \vee \mathcal{A}(m) = \text{undef})}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{A}[m \mapsto \text{undef}], \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, \text{id} \rangle} \text{UNLOCK} \\
\\
\frac{c = \text{unlock}(m) t \in \mathcal{R}. \text{ISR}(t) \neq \text{rpc}(t) = l = \text{ent}_{\mathcal{F}(t)} \mathcal{P}(t) > \mathcal{P}(r) \mathcal{A}(m) \neq \text{tid} = \text{false}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, t, r, ss, \text{id} \rangle} \text{UNLOCK-INT} \\
\\
\frac{c = \text{block } t = r \text{ task}(t) \text{ pc}(t) = l(\text{ss} \vee \text{id}) = \text{true}}{s \Rightarrow_i \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, r, i, ss, \text{id} \rangle} \text{BLK-NS} \\
\\
\frac{c = \text{block } t = r \text{ task}(t) \text{ pc}(t) = l \text{ (ss} \vee \text{id)} = \text{false} \exists ts \in \mathcal{R}. \text{task}(ts) \wedge ts \neq r \wedge \mathcal{P}(ts) = \max(\{\mathcal{P}(u) \mid u \in \mathcal{R} - \{r\} \wedge \text{task}(u)\})}{s \Rightarrow_i \langle \mathcal{B} \cup \{r\}, \mathcal{S}, \mathcal{R} - \{r\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \text{pc}[t \mapsto l'], \phi, ts, i, ss, \text{id} \rangle} \text{BLK-CS}
\end{array}$$

$$\begin{array}{c}
c = \mathbf{start} \quad t = r = 0 \quad (ss \vee id) = \mathbf{false} \quad \exists ts \in (\mathcal{S} \cup \mathcal{R}). \mathit{task}(ts) \wedge \mathcal{P}(ts) = \mathit{max}(\{\mathcal{P}(u) \mid u \in \mathcal{S} \cup \mathcal{R} \wedge \mathit{task}(u)\}) \\
\hline
s \Rightarrow_i \langle \mathcal{B}, \emptyset, \mathcal{S} \cup \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \mathit{pc}[t \mapsto t'], \phi, ts, i, \mathbf{false}, \mathbf{false} \rangle \quad \text{START} \\
\\
t \in \mathcal{B}\mathit{task}(r)((ss \vee id) = \mathbf{true} \vee \mathcal{P}(t) \leq \mathcal{P}(r)) \\
\hline
s \Rightarrow_* \langle \mathcal{B} - \{t\}, \mathcal{S}, \mathcal{R} \cup \{t\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \mathit{pc}, \phi, r, i, ss, id \rangle \quad \text{UNBLK-NS} \\
\\
t \in \mathcal{B}\mathit{task}(r)(ss \vee id) = \mathbf{false} \wedge \mathcal{P}(t) > \mathcal{P}(r) \\
\hline
s \Rightarrow_* \langle \mathcal{B} - \{t\}, \mathcal{S}, \mathcal{R} \cup \{t\}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \mathit{pc}, \phi, t, i, ss, id \rangle \quad \text{UNBLK-CS} \\
\\
t \in \mathcal{R}\mathit{task}(t) \neq r(ss \vee id) = \mathbf{false} \wedge \mathcal{P}(t) = \mathcal{P}(r) \\
\hline
s \Rightarrow_* \langle \mathcal{B}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{A}, \mathcal{F}, \mathit{pc}, \phi, t, i, ss, id \rangle \quad \text{TSHARE}
\end{array}$$

For the commands `skip`, `x:=e`, `assume`, `disableint`, `enableint`, `lock`, and `unlock` permitted in an ISR thread, the following constraints need to hold on  $s'$ . If the current ISR thread is executing the last statement then  $r'$  is the highest priority ISR which was interrupted, if there exists one, and  $i' = i$ . If no ISRs were interrupted then  $r' = i$ , the interrupted task thread and  $i' = \mathit{main}$ , a default value. Also,  $\mathit{pc}'(t) = \mathit{ent}_{\mathcal{F}(t)}$ .

# Synchronization Under Dynamic Constraints

Petra Wolf 

Universität Trier, Fachbereich IV, Informatikwissenschaften, Germany

<https://www.wolfp.net/>

wolfp@informatik.uni-trier.de

---

## Abstract

We introduce a new natural variant of the synchronization problem. Our aim is to model different constraints on the order in which a potential synchronizing word might traverse through the states. We discuss how a word can induce a state-order and examine the computational complexity of different variants of the problem whether an automaton can be synchronized with a word of which the induced order agrees with a given relation. While most of the problems are PSPACE-complete we also observe NP-complete variants and variants solvable in polynomial time. One of them is the careful synchronization problem for partial weakly acyclic automata (which are partial automata whose states can be ordered such that no transition leads to a smaller state), which is shown to be solvable in time  $\mathcal{O}(k^2n^2)$  where  $n$  is the size of the state set and  $k$  is the alphabet-size. The algorithm even computes a synchronizing word as a witness. This is quite surprising as the careful synchronization problem uses to be a hard problem for most classes of automata. We will also observe a drop in the complexity if we track the orders of states on several paths simultaneously instead of tracking the set of active states. Further, we give upper bounds on the length of a synchronizing word depending on the size of the input relation and show that (despite the partiality) the bound of the Černý conjecture also holds for partial weakly acyclic automata.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata theory; Theory of computation → Problems, reductions and completeness

**Keywords and phrases** Synchronizing automaton, Černý conjecture, Reset sequence, Dynamic constraints, Computational complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2020.58

**Related Version** A full version is available at <https://arxiv.org/abs/1910.01935>.

**Funding** Petra Wolf: DFG project FE 560/9-1.

**Acknowledgements** I thank all colleges who commented on drafts of this work, esp. Henning Fernau.

## 1 Introduction

We call  $A = (Q, \Sigma, \delta)$  a deterministic partial (semi-) automaton (DPA) if  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet, and  $\delta: Q \times \Sigma \rightarrow Q$  is a (potentially partial) transition function. If  $\delta$  is defined for every element in  $Q \times \Sigma$ , we call  $A$  a deterministic complete (semi-) automaton (DCA). Clearly, every DCA is also a DPA. We do not specify any start and final states as we are only interested in the transition of states. A DCA  $A = (Q, \Sigma, \delta)$  is *synchronizing* if there exists a word  $w \in \Sigma^*$  such that  $w$  takes every state to the same state. In that case, we call  $w$  a *synchronizing word* for  $A$ . If we are only interested in synchronizing a subset of states  $S \subseteq Q$  we refer to the problem as *subset synchronization*.

One of the oldest applications of the intensively studied topic of synchronizing automata is the problem of designing parts orienters, which are robots or machines that get an object in an (due to a lack of expensive sensors) unknown orientation and transform it into a defined orientation [2]. In his pioneering work, Natarajan [17] modeled the parts orienters as deterministic complete automata where a state corresponds to a possible orientation of a part and a transition of some letter  $a$  from state  $q$  corresponds to applying the modifier



© Petra Wolf;

licensed under Creative Commons License CC-BY

40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020).

Editors: Nitin Saxena and Sunil Simon; Article No. 58; pp. 58:1–58:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

corresponding to  $a$  to a part in orientation  $q$ . He proved that the synchronization problem is solvable in polynomial time for – what is later called – the class of *orientable automata* [20] if the cyclic order respected by the automaton is part of the input. Many different classes of automata have since been studied regarding their synchronization behavior. We refer to [27, 4, 1] for an overview. The original motivation of designing a parts orienter was revisited in [26] where Türker and Yenigün modeled the design of an assembly line, which again brings a part from an unknown orientation into a known orientation, where different modifiers have different costs. What has not been considered so far is that different modifiers can have different impact on the parts and as we do not know the current orientation we might want to restrict the chronology of applied modifiers. For example, if the part is a box with a fold-out lid, turning it upside-down will cause the lid to open. In order to close the lid one might need another modifier such as a low bar which brushes the lid and closes it again. To specify that a parts orienter should deliver the box facing upward with a closed lid one needs to encode something like: “When the box is in the state *facing down*, it later needs to be in the state *lid closed*”. But this does not stop us from opening the lid again, so we need to be more precise and encode: “After **the last time** the box was in the state *facing down*, it needs to visit the state *lid closed* at least once”. We will implement these conditions in our model of a parts orienter by enhancing a given DCA with a relation  $R$ . We will then consider different ways of how a synchronizing word implies an order on the states and ask whether there exists a synchronizing word whose implied state-order agrees with the input-relation  $R$ . The case-example above will be covered by the first two introduced orders. The third considered order relates to the following scenario: Let us again picture the box with the lid in mind, but this time the box initially contains some water. We would like to have the box in a specific orientation with the lid open but the water should not be shed during orientating. We have a modifier that opens the lid and a modifier which rotates the box. Clearly we do not want the box to face downwards after the lid has been opened. So, we encode: “As soon as the state *lid open* has been reached, the state *facing downwards* should never be entered again”.

For every type of dynamic constraint (which we will also call *order*), we investigate the computational complexity of the problem whether a given automaton admits a synchronizing word that transitions the states of the automaton in an order that is conform with a given relation. Thereby, we distinguish between tracking all active states simultaneously and tracking each state individually. We observe different complexities for different ordering concepts and get a good understanding of which ordering constraints yield tractable synchronization problems and which do not. The complexity of the problem also depends on how detailed we describe the allowed sequence of states.

## 2 Related Work

The problem of checking whether a synchronizing word exists for a DCA  $A = (Q, \Sigma, \delta)$  can be solved in time  $\mathcal{O}(|Q|^2|\Sigma|)$ , when no synchronizing word is computed, and in time  $\mathcal{O}(|Q|^3)$  when a witnessing synchronizing word is demanded [11, 27]. In comparison, if we only ask for a subset of states  $S \subseteq Q$  to be synchronized, the problem becomes PSPACE-complete for general DCAs [22]. These two problems have been investigated for several smaller classes of automata involving orders on states. Here, we want to mention the class of *oriented* automata whose states can be arranged in a cyclic order which is preserved by all transitions, which have been studied among others in [17, 11, 2, 21, 27]. If the order is linear instead of cyclic, we get the class of *monotone* automata which has been studied in [2, 21]. An automaton is

called *aperiodic* [4] if there is a non-negative integer  $k$  such that for any word  $w$  and any state  $q$  it holds that  $\delta(q, w^k) = \delta(q, w^{k+1})$ . An automaton is called *weakly acyclic* [20] if there exists an ordering of the states  $q_1, q_2, \dots, q_n$  such that if  $\delta(q_i, a) = q_j$  for some letter  $a \in \Sigma$ , then  $i \leq j$ . In other words, all cycles in a WAA are self-loops. In Section 3 we will consider partial WAAs. The class of WAAs forms a proper subclass of the class of aperiodic automata. Each synchronizing aperiodic automaton admits a synchronizing word of length at most  $n(n-1)/2$  [25], whereas synchronizing WAAs admit synchronizing words of linear lengths [20]. Asking whether an aperiodic automaton admits a synchronizing word of length at most  $k$  is an NP-complete task [27] as it is for general DCAs [18, 11]. The subset synchronization problem for WAAs, and hence for aperiodic automata, is NP-complete [20].

Going from complete automata to partial automata normally brings a jump in complexity. For example, the so called *careful synchronization* problem for DPAs asks for synchronizing a partial automata such that the synchronizing word  $w$  is defined on all states. The problem is PSPACE-complete for DPAs with a binary alphabet [15]. It is even PSPACE-complete for DPAs with a binary alphabet if  $\delta$  is undefined for only one pair in  $Q \times \Sigma$  [16]. The length of a shortest carefully synchronizing word  $c(n)$ , for a DPA with  $|Q| = n$ , differs with  $\Omega(3^{\frac{n}{3}}) \leq c(n) \leq \mathcal{O}(4^{\frac{n}{3}} \cdot n^2)$  [16] significantly from the cubic upper-bound for complete automata. Also for the smaller class of monotone partial automata with an unbounded alphabet size, an exponential lower bound on the length of a shortest carefully synchronizing word is known, while for fixed alphabet sizes of 2 and 3 only a polynomial lower bound is obtained [21]. The careful synchronization problem is NP-hard for partial monotone automata over a four-letter alphabet [26, 21]. It is also NP-hard for aperiodic partial automata over a three-letter alphabet [20]. In contrast we show in Section 3 that the careful synchronization problem is decidable in polynomial time for partial WAAs.

In [20, 21] several hardness and inapproximability results are obtained for WAAs, which can be transferred into our setting as depicted in Section 3. We will also observe W[1]-hardness results from the reductions given in [20]. So far, only little is known (see for example [13, 28, 5]) about the parameterized complexity of all the different synchronization variants considered in the literature.

While synchronizing an automaton under a given order, the set of available (or allowed) transitions per state may depend on the previously visited states on all paths. This dynamic can also be observed in weighted and timed automata [10]. More static constraints given by a second automaton have been discussed in [12]. Due to space limitations missing proofs can be found in the long version of this work [29].

### 3 Problem Definitions

A deterministic semi-automaton  $A = (Q, \Sigma, \delta)$  that might either be partial or complete is called an *automaton*. The transition function  $\delta$  is generalized to words in the usual way. It is further generalized to sets of states  $S \subseteq Q$  as  $\delta(S, w) := \{\delta(q, w) \mid q \in S\}$ . We sometimes refer to  $\delta(S, w)$  as  $S.w$ . We call a state  $q$  *active* regarding a word  $w$  if  $q \in Q.w$ . If for some  $w \in \Sigma^*$ ,  $|Q.w| = 1$  we call  $q \in Q.w$  a *synchronizing state*. We denote by  $|S|$  the size of the set  $S$ . With  $[i..j]$  we refer to the set  $\{k \in \mathbb{N} \mid i \leq k \leq j\}$ . For a word  $w$  over some alphabet  $\Sigma$ , we denote by  $|w|$  the length of  $w$ , by  $w[i]$  the  $i^{\text{th}}$  symbol of  $w$  (or the empty word  $\epsilon$  if  $i = 0$ ) and by  $w[i..j]$  the factor of  $w$  from symbol  $i$  to symbol  $j$ . For each state  $q$ , we call the sequence of active states  $q.w[i]$  for  $0 \leq i < |w|$  the *path* induced by  $w$  starting at  $q$ . We expect the reader to be familiar with basic concepts in complexity theory, approximation theory and parameterized complexity theory [9, 24, 3].

We are now presenting different orders  $\prec_w$  which describe how a word traverses an automaton. We describe how a word implies each of the three presented orders. The first two orders relate the last visits of the states to each other, while the third type of order relates the first visits. We will then combine the order with an automaton  $A$  and a relation  $R \subseteq Q^2$  given in the input and ask whether there exists a synchronizing word for  $A$  such that the implied order of the word agrees with the relation  $R$ . An order  $\prec_w$  agrees with a relation  $R \subseteq Q^2$  if and only if for all pairs  $(p, q) \in R$  it holds that  $p \prec_w q$ , i.e.,  $R \subseteq \prec_w$ .

For any of the below defined orders  $\prec_w \subseteq Q \times Q$ , we define the problem of *synchronization under order* and *subset synchronization under order* as:

► **Definition 1** (SYNC-UNDER- $\prec_w$ ). *Given a DCA  $A = (Q, \Sigma, \delta)$  and a relation  $R \subseteq Q^2$ . Does there exist a word  $w \in \Sigma^*$  such that  $|Q.w| = 1$  and  $R \subseteq \prec_w$ ?*

► **Definition 2** (SUBSET-SYNC-UNDER- $\prec_w$ ). *Given a DCA  $A = (Q, \Sigma, \delta)$ ,  $S \subseteq Q$ , and a relation  $R \subseteq Q^2$ . Is there a word  $w \in \Sigma^*$  with  $|S.w| = 1$  and  $R \subseteq \prec_w$ ?*

It is reasonable to distinguish whether the order should include the initial configuration of the automaton or if it should only describe the consequences of the chosen transitions. In the former case, we refer to the problem as SYNC-UNDER- $0\text{-}\prec_w$  (starting at  $w[0]$ ), in the latter case as SYNC-UNDER- $1\text{-}\prec_w$  (starting at  $w[1]$ ), and if the result holds for both variants, we simply refer to it as SYNC-UNDER- $\prec_w$ . Examples for positive and negative instances of the problem synchronization under order for some discussed variants are illustrated in Figure 1. Let  $\text{first}(q, w, S)$  be the function returning the minimum of positions at which the state  $q$  appears as an active state over all paths induced by  $w$  starting at some state in  $S$ . Accordingly, let  $\text{last}(q, w, S)$  return the maximum of those positions. Note that  $\text{first}(q, w, S) = 0$  for all states  $q \in S$  and  $> 0$  for  $q \in Q \setminus S$ . If  $q$  does not appear on a path induced by  $w$  on  $S$ , then set  $\text{first}(q, w, S) := |w| + 1$  and  $\text{last}(q, w, S) := -1$ . In the SYNC-UNDER- $1\text{-}\prec_w$  problem variant, the occurrence of a state at position 0 is ignored (i.e., if  $q$  occurs only at position 0 while reading  $w$  on  $S$ , then  $\text{last}(q, w, S) = -1$ ). In the following definitions let  $A = (Q, \Sigma, \delta)$  be a DCA and let  $p, q \in Q$ . The following relations  $\prec_w$  are defined for every word  $w \in \Sigma^*$ .

► **Definition 3** (Order  $l < l$  on sets).  $p \prec_{w@S}^{l < l} q \Leftrightarrow \text{last}(p, w, Q) < \text{last}(q, w, Q)$ .

► **Definition 4** (Order  $l \leq l$  on sets).  $p \preceq_{w@S}^{l \leq l} q \Leftrightarrow \text{last}(p, w, Q) \leq \text{last}(q, w, Q)$ .

The second order differs from the first in the sense that  $q$  does not have to appear finally without  $p$ , instead they can disappear simultaneously. Further, note that in comparison with order  $\prec_{w@S}^{l < l}$ , for a pair  $(p, q)$  in order  $\preceq_{w@S}^{l \leq l}$  it is not demanded that  $q$  is active after reading  $w$  up to some position  $i > 0$ . This will make a difference when we later consider the orders on isolated paths rather than on the transition of the whole state set. It can easily be verified that for any word  $w \in \Sigma^*$  and any automaton  $A = (Q, \Sigma, \delta)$  the order  $\preceq_{w@S}^{l \leq l}$  is a proper subset of  $\prec_{w@S}^{l < l}$ . For the order  $\preceq_{w@S}^{l \leq l}$ , it makes no difference whether we take the initial configuration into account since states can disappear simultaneously.

So far, we only introduced orders which consider the set of active states as a whole. It did not matter which active state belongs to which path and a state on a path  $\tau$  could stand in a relation with a state on some other path  $\rho$ . But, in most scenarios the fact that we start with the active state set  $Q$  only models the lack of knowledge about the *actual* current state. In practice only one state  $q$  is active and hence any constraints on the ordering of transitioned states should apply to the path starting at  $q$ . Therefore, we are introducing variants of order 1 and 2 which are defined on paths rather than on series of state sets.

► **Definition 5** (Order  $l < l$  on paths).  $p \prec_{w@p}^{l < l} q \Leftrightarrow \forall r \in Q: \text{last}(p, w, \{r\}) < \text{last}(q, w, \{r\})$ .

► **Definition 6** (Order  $l \leq l$  on paths).  $p \alpha_{w@p}^{l \leq l} q \Leftrightarrow \forall r \in Q: \text{last}(p, w, \{r\}) \leq \text{last}(q, w, \{r\})$ .

The orders  $\alpha_{w@p}^{l < l}$  and  $\alpha_{w@p}^{l \leq l}$  significantly differ since the synchronization problem (starting at position 1) for  $\alpha_{w@p}^{l < l}$  is in NP while it is PSPACE-complete for  $\alpha_{w@p}^{l \leq l}$ .

While the previously defined orders are bringing “positive” constraints to the future transitions of a word, in the sense that the visit of a state  $p$  will demand for a later visit of the state  $q$  (as opening the lid demands closing the lid later in our introductory example), we will now introduce an order which yields “negative” constraints. The third kind of order demands for a pair of states  $(p, q)$  that the (first) visit of the state  $q$  forbids any future visits of the state  $p$  (like do not turn the box after opening the lid). This stands in contrast to the previous orders where we could made up for a “forbidden” visit of the state  $p$  by visiting  $q$  again. The order  $l < f$  will only be considered on paths since when we consider the state set  $Q$ , every pair in  $R$  would already be violated in position 0.

► **Definition 7** (Order  $l < f$  on paths).  $p \alpha_{w@p}^{l < f} q \Leftrightarrow \forall r \in Q: \text{last}(p, w, \{r\}) < \text{first}(q, w, \{r\})$ .

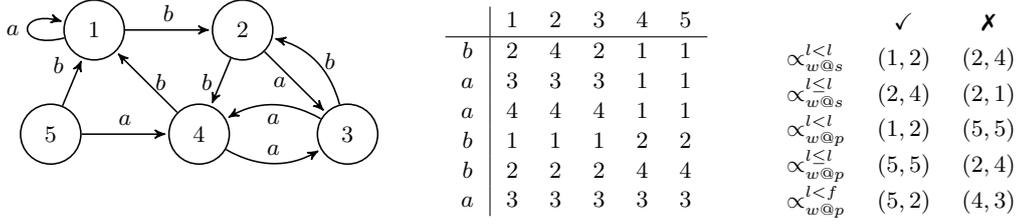
Note that  $\alpha_{w@p}^{l < f}$  is not transitive; e.g., for  $R = \{(1, 2), (2, 3)\}$  we are allowed to go from 3 to 1 as long as we have not transitioned from 1 to 2 yet. For the order  $l < f$ , we will also consider the special case of  $R$  being a strict total order (irreflexive, asymmetric, transitive, and total).

► **Definition 8** (SYNC-UNDER-TOTAL- $\alpha_{w@p}^{l < f}$ ). Given a DCA  $A = (Q, \Sigma, \delta)$ , a strict and total order  $R \subseteq Q^2$ . Is there a word  $w \in \Sigma^*$  with  $|Q.w| = 1$  and  $R \subseteq \alpha_{w@p}^{l < f}$ ?

The orders on path could also be stated as LTL formulas of some kind which need to be satisfied on every path induced by a synchronizing word  $w$  and our hardness results transfer to the more general problem whether a given DCA can be synchronized by a word such that every path induced by  $w$  satisfies a given LTL formula. The orders on sets could be translated into LTL formulas which need to be satisfied on the path in the power-set-automaton starting in the state representing  $Q$ . Despite the similarity of the chosen orders and their translated LTL formulas we need different constructions for the considered orders as the presented attempts mostly do not transfer to the other problems. Therefore, it is not to be expected that a general construction for restricted LTL formulas can be obtained. Our aim is to focus on restricting the order in which states appear and disappear on a path in the automaton or on a path in the power-set-automaton (remember the introductory example). Hence, we have chosen the stated definitions in order to investigate the complexity of problems where the LTL formula is always of the same type, i.e., comparing only the last or first appearances of states on a path. We leave it to future research to investigate other types of LTL formulas. In order to express synchronizability of Kripke structures, an extension to CTL has been introduced in [8]. Note that synchronization of Kripke structures is more similar to D3-directing words [14] for unary NFAs as in contrast to general DFAs the labels of the transitions are omitted in Kripke structures.

► **Definition 9** (CAREFUL SYNC (PSPACE-complete [15])). Given a DPA  $A = (Q, \Sigma, \delta)$ . Is there a word  $w \in \Sigma^*$ , s.t.  $|Q.w| = 1$  and  $w$  is defined on all  $q \in Q$ ?

► **Definition 10** (VERTEX COVER (NP-complete [24])). Given a graph  $G = (V, E)$  and an integer  $k \leq |V|$ . Is there a vertex cover  $V' \subseteq V$  of size  $|V'| \leq k$ ? A vertex cover is a set of states that contains at least one vertex incident to every edge.



■ **Figure 1** DCA  $A$  (left) with all paths induced by  $w = baabba$  (middle) and relations  $R$  consisting of single pairs forming a positive, resp. negative, instance for versions of  $\text{SYNC-UNDER-}<_w$  (right).

■ **Table 1** Overview of the complexity for synchronization (on the left), and subset synchronization under order (on the right) for relations  $\alpha_{w@s}^{l<l}$ ,  $\alpha_{w@p}^{l<l}$ ,  $\alpha_{w@s}^{l\leq l}$ ,  $\alpha_{w@p}^{l\leq l}$ , and  $\alpha_{w@p}^{l<f}$  (tot. is short for total).

Order	Synchronization				Subset Synchronization			
	$l < l$	$l \leq l$	$l < f$	$l < f$ -tot	$l < / \leq l$	$l < f$	$l < f$ -tot	
Set	0	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
	1	PSPACE-c	PSPACE-c	–	–	PSPACE-c	–	–
Path	0	in NP	NP-hard	PSPACE-c	P	PSPACE-c	PSPACE-c	NP-c
	1	in NP	PSPACE-c	PSPACE-c	NP-c	PSPACE-c	PSPACE-c	NP-c

## 4 Main Results

We now investigate the complexity of the introduced problems. An overview on the obtained results is given in Table 1.

► **Theorem 11.** *For all orders  $< \in \{\alpha_{w@s}^{l<l}, \alpha_{w@p}^{l<l}, \alpha_{w@s}^{l\leq l}, \alpha_{w@p}^{l\leq l}, \alpha_{w@p}^{l<f}\}$ , the problem  $\text{SYNC-UNDER-}<$  is contained in PSPACE. Further, it is FPT with parameter  $p = |Q|$ .*

**Proof sketch.** For a DCA  $A = (Q, \Sigma, \delta)$  and  $R \subseteq Q^2$ , we can enrich the powerset-automaton  $\mathcal{P}(A)$  of  $A$  with the information about the set of active pairs in  $R$  in every state. Here, a pair in  $R$  is active during the transition of a word if it constrains which states might be, or need to be visited in the future. For instance, in the example in Figure 1 concerning the order  $\alpha_{w@s}^{l<l}$ , the pair  $(2, 4)$  is active while reading the prefix  $ba$ , since the state 2 has appeared as an active state while the state 4 has not appeared without 2 as an active state yet. It is not active after reading  $baa$ , since now 4 is active without 2 and hence the pair  $(2, 4)$  is satisfied and does not demand for further state visits. The pair becomes active again after reading  $baab$  since again 2 became active demanding for the state 4 to become active without 2 again.

To store the information of active pairs in  $R$ , we copy each state in  $\mathcal{P}(A)$   $2^{|R|}$  times for the orders  $\alpha_{w@s}^{l<l}$ , and  $\alpha_{w@s}^{l\leq l}$ , and  $|Q|2^{|R|}$  times for the orders  $\alpha_{w@p}^{l<l}$ ,  $\alpha_{w@p}^{l\leq l}$ , and  $\alpha_{w@p}^{l<f}$ , yielding an automaton  $\mathcal{P}(A)$  of size at most  $2^{|Q|}|Q|2^{|R|} = 2^{\mathcal{O}(|Q|^2)}$ . As each state contains only up to  $|Q| + 1$  bit-strings of length up to  $|Q|^2$  a state of  $\mathcal{P}(A)$  can be stored in polynomial space. Hence, reachability tests can be performed in  $\mathcal{P}(A)$  in polynomial space and in time  $2^{\mathcal{O}(|Q|^2)}$ . ◀

► **Theorem 12.**  *$\text{SYNC-UNDER-}\alpha_{w@s}^{l\leq l}$  is PSPACE-complete, even for  $|R| = 1$  and  $|\Sigma| = 2$ .*

**Proof.** We reduce from the PSPACE-complete CAREFUL SYNC problem for DPAs [15]. Let  $A = (Q, \Sigma, \delta)$  be a DPA. We construct from  $A$  a DCA  $A' = (Q' = Q \cup \{q_\ominus, r\}, \Sigma, \delta')$  with  $q_\ominus, r \notin Q$ . For every pair  $q \in Q, \sigma \in \Sigma$  for which  $\delta(q, \sigma)$  is undefined, we define the transition  $\delta'(q, \sigma) = q_\ominus$ . On all other pairs  $\delta'$  agrees with  $\delta$ . Further, for some arbitrary state  $t \in Q$  and

for all  $\gamma \in \Sigma$  we set  $\delta'(q_\ominus, \gamma) = \delta'(t, \gamma)$  (note that this can be  $q_\ominus$  itself) and  $\delta'(r, \gamma) = \delta'(t, \gamma)$ . We set the relation  $R$  to  $R := \{(q_\ominus, r)\}$ .

Assume there exists a word  $w \in \Sigma^*$ ,  $|w| = n$  that synchronizes  $A$  without using an undefined transition. Then,  $\delta(q, w[1])$  is defined for all states  $q \in Q$ . The letter  $w[1]$  acts on  $A'$  in the following way: (1)  $\delta'(r, w[1]) = \delta'(q_\ominus, w[1]) = \delta(t, w[1])$  which is defined by assumption; (2)  $\delta'(Q, w[1]) \subseteq Q$  since  $\delta(q, w[1])$  is defined for all states  $q \in Q$ . The combination of (1)-(2) yields  $\delta'(Q', w[1]) \subseteq Q$ . We further constructed  $\delta'$  such that  $\delta'(Q', w[1]) = \delta(Q, w[1])$ . Since  $\delta(q, w[2..n])$  is defined by assumption for every  $q \in \delta(Q, w[1])$ ,  $\delta'$  agrees with  $\delta$  on  $w[2..n]$  for every  $q \in \delta(Q, w[1])$ . This means especially that while reading  $w[2..n]$  in  $A'$  on the states in  $\delta'(Q', w[1])$  the state  $q_\ominus$  is not reached and that  $\delta'(Q', w) = \delta(Q, w)$ . Therefore,  $w$  also synchronizes the automaton  $A'$ . The state  $q_\ominus$  is only active in the start configuration where no letter of  $w$  is read yet and is not active anymore while reading  $w$ . The same holds for  $r$ , hence  $R = \{(q_\ominus, r)\} \subseteq \alpha_{w@_s}^{l < l}$ .

For the other direction, assume there exists a word  $w \in \Sigma^*$ ,  $|w| = n$  that synchronizes  $A'$  with  $(q_\ominus, r) \in \alpha_{w@_s}^{l < l}$ . Then,  $w$  can be partitioned into  $w = uv$  with  $u, v \in \Sigma^*$  where  $r$  is not active while reading the factor  $v$  in  $w$ . The only position of  $w$  in which  $r$  is active due to the definition of  $\delta'$  is before any letter of  $w$  is read. Hence, we can set  $u = \epsilon$  and  $v = w$ . As  $(q_\ominus, r) \in \alpha_{w@_s}^{l < l}$  it holds for all  $i \in [1..n]$  that  $q_\ominus \notin \delta'(Q', v[1..i])$ . Hence,  $\delta'(q, v)$  is defined for every state  $q \in Q$ . Since  $\delta'$  and  $\delta$  agree on the definition range of  $\delta$  it follows that  $v$  also synchronizes the state set  $Q$  in  $A$  without using an undefined transition. ◀

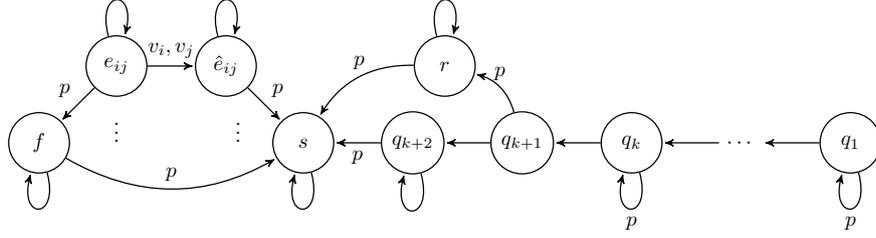
► **Remark 13.** The construction works for both variants (with and without 0) of the problem. It can further be adapted for the order  $\alpha_{w@_s}^{l < l}$  (both variants) by introducing a copy  $\hat{q}$  of every state in  $Q \cup \{r\}$  and setting  $\delta'(\hat{q}, \sigma) = q$  for every  $\sigma \in \Sigma$ ,  $q \in Q \cup \{r\}$ . For all other transitions, we follow the above construction. We keep  $R := \{(q_\ominus, r)\}$ . Since  $r$  is left after  $w[2]$  for any word  $w \in \Sigma^*$  with  $|w| \geq 2$  in order to satisfy  $R$  the state  $q_\ominus$  needs to be left with  $w[1]$  such that afterwards  $r$  is active without  $q_\ominus$ . Note that  $q_\ominus$  has not been copied.

► **Corollary 14.** *SYNC-UNDER- $\alpha_{w@_s}^{l < l}$  is PSPACE-complete even for  $|R| = 1$  and  $|\Sigma| = 2$ .*

► **Remark 15.** The reduction presented in the proof of Theorem 12 can also be applied to show the PSPACE-completeness of SYNC-UNDER-1- $\alpha_{w@_p}^{l < l}$ . Since the state  $r$  cannot be reached from any other state, the state  $q_\ominus$  needs to be left with the first letter of any synchronizing word and must not become active again on any path. The rest of the argument follows the proof of Theorem 12. Note that the construction only works for SYNC-UNDER-1- $\alpha_{w@_p}^{l < l}$ . If we consider SYNC-UNDER-0- $\alpha_{w@_p}^{l < l}$  the problem might become easier. But it is at least NP-hard.

► **Theorem 16.** *The problem SYNC-UNDER-0- $\alpha_{w@_p}^{l < l}$  is NP-hard.*

**Proof.** We give a reduction from VERTEX COVER. We refer to Figure 2 for a schematic illustration. Let  $G(V, E)$  be a graph and let  $k \in \mathbb{N}$ . We construct from  $G$  a DCA  $A = (Q, \Sigma, \delta)$  in the following way. We set  $\Sigma = V \cup \{p\}$  for some  $p \notin V$ . We start with  $Q = \{f, r, s\}$  where  $s$  is a sink state, meaning  $\delta(s, \sigma) = s$  for all  $\sigma \in \Sigma$ ,  $f$  will be the “false way” and  $r$  will be the “right way”. We set  $\delta(r, p) = \delta(f, p) = s$  and  $\delta(r, v) = r$ ,  $\delta(f, v) = f$  for all other  $v \in \Sigma$ . For every edge  $e_{ij} \in E$  connecting some vertices  $v_i, v_j \in V$ , we create two states  $e_{ij}$  and  $\hat{e}_{ij}$  and set  $\delta(e_{ij}, v_i) = \delta(e_{ij}, v_j) = \hat{e}_{ij}$ ,  $\delta(e_{ij}, p) = f$ . For all other letters, we stay in  $e_{ij}$ . For the state  $\hat{e}_{ij}$ , we stay in  $\hat{e}_{ij}$  for all letters except  $p$ . For  $p$ , we set  $\delta(\hat{e}_{ij}, p) = s$ . We further create for  $1 \leq i \leq k + 2$  the states  $q_i$  with the transitions  $\delta(q_i, v) = q_{i+1}$  for  $i \leq k + 1$  and  $v \in V$ ,  $\delta(q_i, p) = q_i$  for  $i \leq k$ , and  $\delta(q_{k+1}, p) = r$ ,  $\delta(q_{k+2}, p) = s$ ,  $\delta(q_{k+2}, v) = q_{k+2}$  for  $v \in V$ . We set  $R := (q_1, r) \cup \{(e_{ij}, \hat{e}_{ij}) \mid e_{ij} \in E\}$ .



■ **Figure 2** Schematic illustration of the reduction from VERTEX COVER (see Theorem 16). For each state, the transition without a label represents all letters which are not explicitly listed as an outgoing transition from that state.

If there exists a vertex cover of size  $k' < k$  for  $G$ , then there also exists a vertex cover of size  $k$  for  $G$ . Therefore, assume  $V'$  is a vertex cover for  $G$  of size  $k$ . Then, the word  $wpp$  where  $w$  is any non-repeating listing of the vertices in  $V'$  is a synchronizing word for  $A$  with  $R \subseteq \alpha_{wpp@p}^{l < l}$ . Since  $q_1$  cannot be reached from any other state, the pair  $(q_1, r) \in R$  is trivially satisfied for each path starting in any state other than  $q_1$ . Hence, we only have to track the appearances of  $q_1$  and  $r$  on the path starting in  $q_1$ . Since  $w$  lists the states in the vertex cover  $V'$  it holds that  $|w| = k$  and hence  $q_1.w = q_{k+1}$ . Further,  $q_1.wp = r$  and  $q_1.wpp = s$ . Hence, the pair  $(q_1, r)$  is satisfied on the path starting in  $q_1$  as well as on all paths. It remains to show that  $wpp$  is indeed a synchronizing word and that all pairs in  $R$  of the form  $(e_{ij}, \hat{e}_{ij})$  are satisfied. For every state  $e_{ij}$  representing an edge  $e_{ij}$ , the state  $\hat{e}_{ij}$  is reached if we read a letter corresponding to a vertex incident to it. Since  $V'$  is a vertex cover, the word  $w$  contains for each edge  $e_{ij}$  at least one vertex incident to it. Hence, for each edge  $e_{ij}.w = \hat{e}_{ij}$  and  $e_{ij}.wpp = s$ . Since each state  $e_{ij}$  is not reachable from any other state it follows that all pairs  $(e_{ij}, \hat{e}_{ij})$  are satisfied by  $wpp$  on all paths. It is easy to see that for all other states  $q \in Q$  it holds that  $q.wpp = s$ .

For the other direction, assume there exists a synchronizing word  $w$  for  $A$  with  $R \subseteq \alpha_{w@p}^{l < l}$ . By the construction of  $A$  the word  $w$  must contain some letters  $p$ . Partition  $w$  into  $w = upv$  where  $p$  does not appear in  $u$ . Since  $R \subseteq \alpha_{w@p}^{l < l}$  the pair  $(q_1, r)$  in  $R$  enforces  $|u| \leq k$  since otherwise the only path on which  $q_1$  appears (namely the one starting in  $q_1$ ) will not contain the state  $r$  as for any longer prefix  $u$  it holds that  $q_1.u = q_{k+2}$  and  $r$  is not reachable from  $q_{k+2}$ . The other pairs of the form  $(e_{ij}, \hat{e}_{ij}) \in R$  enforces that  $u$  encodes a vertex cover for  $G$ . Assume this is not the case, then there is some state  $e_{ij}$  for which  $e_{ij}.u = e_{ij}$ . But then,  $e_{ij}.up = f$  and from  $f$  the state  $\hat{e}_{ij}$  is not reachable, hence the pair  $(e_{ij}, \hat{e}_{ij})$  is not satisfied on the path starting in  $e_{ij}$ . Therefore,  $u$  encodes a vertex cover of size at most  $k$ . ◀

If we consider  $\alpha_{w@p}^{l < l}$ , the two variants of the order (with and without position  $i = 0$ ) do not differ since for a pair  $(p, q)$ , regardless of whether  $p$  is reached, the state  $q$  must be reached on every path. Hence, whenever we leave  $q$  we must be able to return to it, so it does not matter if we consider starting in  $q$  or not. In comparison with SYNC-UNDER-1- $\alpha_{w@p}^{l < l}$ , the problem SYNC-UNDER- $\alpha_{w@p}^{l < l}$  is solvable in polynomial time using non-determinism.

► **Theorem 17.** *The problem SYNC-UNDER- $\alpha_{w@p}^{l < l}$  is in NP.*

**Proof.** Recall that in the problem SYNC-UNDER- $\alpha_{w@p}^{l < l}$ , for every pair of states  $(p, q) \in R$  and every state  $r \in Q$ , it is demanded that  $q$  appears somewhere on a path induced by the sought synchronizing word  $w$ , starting in  $r$ . Hence, a precondition for the existence of  $w$  is that for every pair  $(p_i, q_i) \in R$  the states  $q_i$  must be reachable from any state in  $Q$ . More

precisely, under the order  $\alpha_{w@p}^{l<l}$  only the last appearance of each state on a path is taken into account. Hence, a prohibited visit of a state can later be compensated by revisiting all related states in the correct order. Thus, it is sufficient to first synchronize all pairs of states and then transition the remaining state through all related states in the demanded order. ◀

► **Remark 18.** The NP-hardness proof for SYNC-UNDER- $\theta$ - $\alpha_{w@p}^{l<l}$  in Theorem 16 and the NP-membership proof for SYNC-UNDER- $\alpha_{w@p}^{l<l}$  in Theorem 17 do not work for the respectively other problem since concerning  $\alpha_{w@p}^{l<l}$  the larger states need to be reached on *every* path and not only on a path containing the corresponding smaller state as it is the case concerning  $\alpha_{w@p}^{l\leq l}$ .

► **Theorem 19.** *The problem SYNC-UNDER- $\theta$ - $\alpha_{w@p}^{l<f}$  is PSPACE-complete.*

**Proof sketch.** We reduce from CAREFUL SYNC. As in the proof of Theorem 12 we take every undefined transition  $\delta(q, \sigma)$  to the new state  $q_\ominus$ . We further enrich the alphabet by a letter  $c$  and use  $c$  to take  $q_\ominus$  into  $Q$ . We use the relation  $R$  and extra states  $r, s$  to enforce that  $c$  is the first letter of any synchronizing word, and that afterwards  $q_\ominus$  is not reached again. ◀

► **Remark 20.** In the presented way, the reduction relies on taking the initial configuration at position  $i = 0$  into account but we can adapt the construction to prove PSPACE-completeness of SYNC-UNDER- $1$ - $\alpha_{w@p}^{l<f}$  by copying every state in  $Q$  and the state  $r$ . Denote a copy of a state  $q$  with  $q'$ . Then, we set  $\delta'(q', \sigma) = q$  for any copied state including  $r'$ . Note that the copied states are not reachable from any state. Now, after the first transition  $w[1]$  (which can be arbitrary), we have a similar situation as previously considered for  $w[0]$ . The state  $r$  is active and forces the next letter to be the letter  $c$ ; all states in  $Q$  are active; reading the letter  $c$  will cause all states  $q_\sigma$  to be left and never be reached again.

In the above reduction from CAREFUL SYNC the size of  $R$  depends on  $|Q|$ . Hence, the question whether SYNC-UNDER- $\alpha_{w@p}^{l<f}$  is PSPACE-hard for  $|R| = 1$  is an interesting topic for further research. We will now see that when  $R$  is a strict and total order on  $Q$ , the problem of synchronizing under  $\alpha_{w@p}^{l<f}$  (a.k.a. SYNC-UNDER-TOTAL- $\alpha_{w@p}^{l<f}$ ) becomes tractable.

► **Theorem 21.** *Let  $A = (Q, \Sigma, \delta)$ ,  $R$  be an instance of SYNC-UNDER-TOTAL- $\alpha_{w@p}^{l<f}$ . A shortest synchronizing word  $w$  for  $A$  with  $R \subseteq \alpha_{w@p}^{l<f}$  has length  $|w| \leq \frac{|Q|(|Q|-1)}{2} + 1$ .*

Note that this length bound is smaller than the bound of the Černý conjecture [6, 7] for  $|Q| > 3$ . The same bound can be obtained for SUBSET-SYNC-UNDER-TOTAL- $\alpha_{w@p}^{l<f}$ . We will now prove that the problem SYNC-UNDER-TOTAL- $\theta$ - $\alpha_{w@p}^{l<f}$  is equivalent – concerning polynomial time many-one-reductions (depicted by  $\equiv_p$ ) – to the problem of carefully synchronizing a partial weakly acyclic automaton (PWAA) (a PWAA is a WAA where  $\delta$  might be only partially defined). The obtained length bound also holds for PWAAs, which is only a quadratic increase w.r.t. the linear length bound in the complete case [20]. This is quite surprising as in general shortest carefully synchronizing words have an exponential lower bound [16]. Further, we show that careful synchronization for PWAAs is in P while the problem is PSPACE-complete for general DPAs even if only one transition is undefined [16].

► **Theorem 22.** *SYNC-UNDER-TOTAL- $\theta$ - $\alpha_{w@p}^{l<f} \equiv_p$  CAREFUL SYNC of PWAAs.*

**Proof.** We prove this statement by reducing the two problems to each other. Let  $A = (Q, \Sigma, \delta)$ ,  $R \subseteq Q^2$  be an instance of SYNC-UNDER-TOTAL- $\theta$ - $\alpha_{w@p}^{l<f}$ . Since  $R$  is a strict total order on  $Q$ , we can order the states according to  $R$ . We construct from  $A$  the PWAA  $A' = (Q, \Sigma, \delta')$  by removing all transitions in  $\delta$  which are leading backwards in the order. Clearly,  $A'$  is carefully synchronizable if and only if  $A$  is synchronizable with respect to  $R$ .

For the other reduction, assume  $A = (Q, \Sigma, \delta)$  is a PWAA. Then, we can order the states in  $Q$  such that no transition leads to a smaller state. We are constructing from  $A$  the DCA  $A' = (Q \cup \{q_<\}, \Sigma, \delta')$  and insert  $q_<$  as the smallest state in the state ordering. Then, we define in  $\delta'$  all transitions  $(q, \sigma)$  for  $q \in Q, \sigma \in \Sigma$  which are undefined in  $\delta$  as  $\delta'(q, \sigma) = q_<$ . We take the state  $q_<$  with every symbol to the maximal state in the order. Note that the maximal state needs to be the synchronizing state if one exists. We set  $R = \{(p, q) \mid p < q \text{ in the state ordering of } Q \text{ in } A\} \cup \{(q_<, q) \mid q \in Q\}$ . Every undefined transition  $(p, \sigma)$  in  $A$  is not allowed in  $A'$  at any time, since otherwise the pair  $(q_<, p) \in R$  would be violated. The state  $q_<$  itself can reach the synchronizing state with any transition. Hence,  $A'$  is synchronizable with respect to  $R$  if and only if  $A$  is carefully synchronizable. ◀

► **Theorem 23.** *Let  $A = (Q, \Sigma, \delta)$ ,  $R \subseteq Q^2$  with  $n := |Q|$ . Let  $Q_1 \subseteq Q$  be such that  $R$  restricted to  $Q_1 \times Q_1$  is a strict and total order. Let  $p = |Q| - |Q_1|$ . For  $\text{SYNC-UNDER-}\alpha_{w@p}^{l<f}$ : If  $A$  is synchronizable by a shortest word  $w$  with  $R \subseteq \alpha_{w@p}^{l<f}$ , then:  $|w| \leq \binom{n(n-1)}{2} + 1 \cdot 2^p$ .*

We now present an  $\mathcal{O}(|\Sigma|^2|Q|^2)$  algorithm for  $\text{SYNC-UNDER-TOTAL-}\alpha_{w@p}^{l<f}$ . The idea is the following: We start on all states as the set of active states and pick a letter, which is defined on all active states and maps at least one active state to a larger state in the order  $R$ . We collect the sequence  $u$  of applied letters and after each step, we apply the whole sequence  $u$  on the set of active states. This is possible as we already know that  $u$  is defined on  $Q$ . We thereby ensure that a state which has become inactive after some iteration never becomes active again after an iteration step and hence  $\Sigma_{\text{def}}$  grows in each step and never shrinks. While a greedy algorithm which does not store  $u$  runs in  $\mathcal{O}(|\Sigma||Q|^3)$ , with this trick we get a running time of  $\mathcal{O}(|\Sigma|^2|Q|^2)$ . As in practice  $|Q| \gg |\Sigma|$  this is a remarkable improvement. Note that we can store  $u$  compactly by only keeping the map induced by the current  $u$  and storing the sequence of letters  $\sigma$  from which we can restore the value of  $u$  in each iteration.

► **Theorem 24.**  *$\text{SYNC-UNDER-TOTAL-}\alpha_{w@p}^{l<f}$  is solvable in quadratic time.*

**Proof.** Let  $A = (Q, \Sigma, \delta)$  be a DCA, and let  $R \subseteq Q^2$  be a strict and total order on  $Q$ . Figure 3 describes an algorithm that decides in time  $\mathcal{O}(|\Sigma|^2|Q|^2)$  whether  $A$  is synchronizable with respect to  $R$  under the order  $\alpha_{w@p}^{l<f}$  (including position  $i = 0$ ) on paths. Despite the simplicity of the algorithm its correctness is not trivial and is proven in the following lemmas. ◀

► **Lemma 25.** *The algorithm in Figure 3 terminates on every input  $A = (Q, \Sigma, \delta)$  with  $m := |\Sigma|$ ,  $n := |Q|$ , strict and total order  $R \subseteq |Q|^2$  in time  $\mathcal{O}(m^2n^2)$ .*

**Proof.** Step 1 can be performed in time  $\mathcal{O}(n \log n)$  using the Quicksort-algorithm. Step 2 to Step 5 take time  $\mathcal{O}(mn)$  each. The procedure **explore** takes time  $\mathcal{O}(mn^2)$ . The number of iterations in Step 6 is bounded by  $|\Sigma_{\text{part}}|$  as  $\Sigma_{\text{def}}$  is applied exhaustively on  $Q_{\text{act}}$  and by invariant (2) of Lemma 26 we have  $Q'_{\text{act}} \subseteq Q_{\text{act}}$ . This yields a total run-time of  $\mathcal{O}(m^2n^2)$ . ◀

► **Lemma 26.** *If the algorithm in Figure 3 returns true on the input  $A = (Q, \Sigma, \delta)$ , strict and total order  $R \subseteq |Q|^2$ , then  $A, R$  is a yes instance of  $\text{SYNC-UNDER-TOTAL-}\alpha_{w@p}^{l<f}$ .*

**Proof.** For the procedure **explore**, the following invariant holds: Let  $u_{\text{old}}$  be the word  $u$  before the execution of **explore** and let  $u_{\text{new}}$  be the one after the execution of **explore**. Then, it holds for all executions of **explore** that (1)  $Q.u_{\text{new}}$  is defined, (2)  $Q.u_{\text{new}} \subseteq Q.u_{\text{old}}$ , and (3)  $Q.u_{\text{new}}u_{\text{new}} = Q.u_{\text{new}}$ . We prove the invariant by induction. First, note that the word  $u$  computed by **explore** in Step 5 is defined on all states in  $Q$  since it only consists of letters which are defined on all states. Since we go through the states in order during the

**Step 1:** Order all states in  $Q$  according to the order  $R$ . Since  $R$  is strict and total the states can be ordered in an array  $\{q_1, q_2, \dots, q_n\}$ .

**Step 2:** Delete in the automaton  $A$  all transitions which are leading backwards in the state-ordering. If this produces a state with no outgoing arc, abort; return false.

**Step 3:** Let  $q_n$  be the maximal state according to the order  $R$ . Delete all transitions in  $A$  which are labeled with letters  $\sigma \in \Sigma$  for which  $q_n.\sigma$  is undefined. If this produces a state with no outgoing transition, abort and return false.

**Step 4:** Partition the alphabet  $\Sigma$  into  $\Sigma_{\text{def}}$ , consisting of all letters  $\sigma \in \Sigma$  for which  $q.\sigma$  is defined for all states  $q \in Q$ , and  $\Sigma_{\text{par}} := \Sigma \setminus \Sigma_{\text{def}}$ . If  $\Sigma_{\text{def}} = \emptyset$  abort; return false.

**Step 5:** Compute  $\text{explore}(Q, Q, \Sigma_{\text{def}}, \epsilon)$  which returns  $Q_{\text{act}}$  and  $u \in \Sigma_{\text{def}}^*$ . The returned set of active states will equate  $Q_{\text{trap}} = \{q \in Q \mid q.\Sigma_{\text{def}} = q\}$ .

**Step 6:** Set  $\Sigma_{\text{def}} := \Sigma_{\text{def}} \cup \{\sigma \in \Sigma_{\text{par}} \mid q.\sigma \text{ is defined for all } q \in Q_{\text{act}}\}$ . Compute  $\text{explore}(Q, Q_{\text{act}}, \Sigma_{\text{def}}, u)$  which returns  $Q'_{\text{act}}$  and  $u' \in \Sigma_{\text{def}}^*$ . Set  $Q_{\text{act}} := Q'_{\text{act}}$ ,  $u := u'$ ,  $\Sigma_{\text{par}} := \Sigma \setminus \Sigma_{\text{def}}$ . Repeat this step until  $Q_{\text{act}}$  does not change anymore ( $\equiv$  to  $\Sigma_{\text{def}}$  does not change anymore). Then, if  $Q_{\text{act}} = \{q_n\}$  return true, otherwise return false.

**Procedure explore:** Input: Ordered state set  $Q$ , set of active states  $Q_{\text{act}}$ , alphabet  $\Sigma_{\text{exp}}$  to be explored, word  $u$  with  $Q.u = Q_{\text{act}}$ . Initialize a new word  $u' := u$ . Go through the active states in order. For the current state  $q$ , test if any  $\sigma \in \Sigma_{\text{def}}$  leads to a larger state, if so, perform the transition  $\sigma u$  on all active states and update the set of active states  $Q_{\text{act}}$ . Concatenate  $u'$  with  $\sigma u$ . Continue with the next larger active state (note that this can be  $q.\sigma u$ ). If  $q_n$  is reached, return  $u'$ , and the current set of active states  $Q_{\text{act}}$ .

■ **Figure 3** Poly-time algorithm for SYNC-UNDER-TOTAL- $\theta$ - $\alpha_{w@p}^{l < f}$  on  $A = (Q, \Sigma, \delta)$ ,  $R \subseteq Q^2$ .

execution of **explore** and we only proceed with the next larger state if (1) we were able to leave the current one towards a larger state or if (2) the current state cannot be left with any of the letters in  $\Sigma_{\text{def}}$ , it holds that  $Q.uu = Q.u$ . Also, trivially  $Q.u \subseteq Q$ .

Next, consider some later execution of **explore**. The new word computed by **explore** is of the form  $u_{\text{new}} := u_{\text{old}}\sigma_1 u_{\text{old}}\sigma_2 u_{\text{old}} \dots \sigma_i u_{\text{old}}$  for some  $0 \leq i \leq |Q|$ . The induction hypothesis tells us that (1)  $Q.u_{\text{old}}$  is defined. Since  $Q.u_{\text{old}}\sigma_1$  is defined (since  $\sigma_1 \in \Sigma_{\text{def}}$ ) and  $Q.u_{\text{old}}\sigma_1 \subseteq Q$  it holds that  $Q.u_{\text{old}}\sigma_1 u_{\text{old}}$  is defined. Further, since  $u_{\text{old}}$  brings all states to the set  $Q.u_{\text{old}}$  it also brings a subset of  $Q$  to a subset of  $Q.u_{\text{old}}$ . Using the induction hypothesis (3) we get by an induction on  $i$  that  $Q.u_{\text{new}}$  is defined and  $Q.u_{\text{new}} \subseteq Q.u_{\text{old}}$ . Since in the execution of **explore** we only proceed with the next larger state if we exhaustively checked all possible transitions for the current state and since  $Q.u_{\text{old}}u_{\text{old}} = Q.u_{\text{old}}$  it follows that  $Q.u_{\text{new}}u_{\text{new}} = Q.u_{\text{new}}$ .

If the algorithm in the proof of Theorem 24 terminates and returns yes, it also returns a synchronizing word  $u$ . By the invariant proven above, we know that  $Q.u$  is defined. This means that  $u$  never causes a transition of a larger state to a smaller state and hence  $\alpha_{u@p}^{l < f}$  agrees with  $R$ . During the execution of the algorithm we track the set of active states  $Q_{\text{act}}$  (starting with  $Q$ ) and only return true if  $Q_{\text{act}}$  contains only the in  $R$  largest state  $q_n$ . Since  $R$  is a total order, every  $q \in Q$  is smaller than  $q_n$  and hence  $q_n$  cannot be left. Therefore,  $q_n$  needs to be the single synchronizing state of  $A$  and  $u$  is a synchronizing word for  $A$ . ◀

► **Lemma 27.** *If the algorithm in Figure 3 returns false on input  $A = (Q, \Sigma, \delta)$ , strict and total order  $R \subseteq |Q|^2$ , then  $A$  is not synchronizable with respect to  $R$  under the order  $\alpha_{w@p}^{l < f}$ .*

**Proof.** The algorithm returns false in the following cases.

(1) All outgoing transitions of some state  $q$  are deleted in Step 2. In that case, every transition of  $q$  leads to a smaller state. As this would violate the order  $R$ , we cannot perform any of those transitions. Hence,  $q$  cannot be left. (The case that  $q = q_n$  is treated in (2).)

(2) Since  $q_n$  is the largest state, it cannot be left. Hence,  $q_n$  will be active the whole time. Therefore, any transition which is not defined for  $q_n$  cannot be taken at all since  $q_n$  is active during the whole synchronizing process. Hence, we can delete these transitions globally. If this creates a state which cannot be left anymore, this state cannot be synchronized.

(3) The execution of **explore** returns two identical sets of active states  $Q_{\text{act}}$  in a row. Let  $\Sigma_{\text{def}}$  be the explored alphabet of the last execution of **explore**. Then,  $\Sigma_{\text{def}}$  contains all letters  $\sigma$  from  $\Sigma$  for which  $q.\sigma$  is defined on all states  $q \in Q_{\text{act}}$  and none of them leads some state in  $Q_{\text{act}}$  to a larger state. Since the relation  $R$  forbids cycles, for all  $\sigma \in \Sigma_{\text{def}}$  and all  $q \in Q_{\text{act}}$   $q.\sigma = q$  and hence this set cannot be left when all states of the set are active simultaneously. Since all states are active at the beginning of the algorithm, also all states in  $Q_{\text{act}}$  are active and since this set cannot be left with any transition which does not cause an undefined transition for all states in the set, the state set cannot be synchronized at all. ◀

► **Corollary 28.** *The careful synchronization problem for PWAA is in P.*

If we allow one unrestricted transition first (SYNC-UNDER-TOTAL-1- $\alpha_{w@p}^{l < f}$ ) the problem is related to the subset synchronization problem of complete WAAs which is NP-complete [20]. Together with the quadratic length bound of a synchronizing word of SYNC-UNDER-TOTAL-1- $\alpha_{w@p}^{l < f}$  (which implies membership of SYNC-UNDER-TOTAL-1- $\alpha_{w@p}^{l < f}$  in NP), we get:

► **Theorem 29.** *The problem SYNC-UNDER-TOTAL-1- $\alpha_{w@p}^{l < f}$  is NP-complete.*

**Proof sketch.** We reduce from the NP-complete problem: Given a complete weakly acyclic automaton  $A = (Q, \Sigma, \delta)$  and a subset  $S \subseteq Q$ , does there exist word  $w \in \Sigma^*$  such that  $|S.w| = 1$ ? We construct from  $A$  an automaton  $A' = (Q', \Sigma \cup \{c\}, \delta')$  with  $c \notin \Sigma$  in the following way. We start with  $Q' = Q$ . W.l.o.g., assume  $|S| \geq 2$ . For each state  $q \in S$ , we add a copy  $\hat{q}$  to  $Q'$ . Further, we add the states  $q_<$  and  $q_>$ . Let  $q_1, q_2, \dots, q_n$  be an ordering of the states in  $Q$  such that  $\delta$  follows this ordering. The transition function  $\delta'$  agrees with  $\delta$  on all states in  $Q$  and letters in  $\Sigma$ . For a copied state  $\hat{q}$ , we set  $\delta'(\hat{q}, \sigma) = \hat{q}$  for all  $\sigma \in \Sigma$  and  $\delta'(\hat{q}, c) = q$ . For every state  $q \in Q$ , we set  $\delta'(q, c) = q_<$ . Let  $q_s$  be some state in  $S$ . Then for all  $\sigma \in \Sigma$  we set  $\delta'(q_<, \sigma) = \delta(q_s, \sigma)$ ,  $\delta'(q_<, c) = q_s$  and  $\delta'(q_>, \sigma) = q_>$ ,  $\delta'(q_>, c) = q_s$ . Then, we set  $R = \{(q_i, q_j) \mid i < j\}$  for all states in  $Q$ . Further, for every copied state  $\hat{q}_k$  we extend  $R$  by the sets:  $\{(\hat{q}_k, q_k)\}$ ,  $\{(q_i, \hat{q}_k), (\hat{q}_k, q_j) \mid i < k, k < j\}$ , and  $\{(\hat{q}_i, \hat{q}_k), (\hat{q}_k, \hat{q}_j) \mid i < k < j\}$  for all copied states  $\hat{q}_i, \hat{q}_j$ . For the states  $q_<, q_>$ , we add  $\{(q_<, q) \mid q \neq q_< \in Q'\}$  and  $\{(q, q_>) \mid q \neq q_> \in Q'\}$  to  $R$ . ◀

► **Theorem 30.** *SUBSET-SYNC-UNDER-TOTAL- $\alpha_{w@p}^{l < f}$  is NP-complete.*

► **Theorem 31.** *The following subset synchronization problems are PSPACE-complete for both -0- and -1-: SUBSET-SYNC-UNDER- $\alpha_{w@s}^{l < l}$ ,  $-\alpha_{w@p}^{l < l}$ ,  $-\alpha_{w@s}^{l < l}$ ,  $-\alpha_{w@p}^{l < l}$ ,  $-\alpha_{w@s}^{l < f}$ .*

Several other results can be transferred from [20] to the corresponding version of the SYNC-UNDER-TOTAL-0- $\alpha_{w@p}^{l < f}$  problem, such as inapproximability of the problems of finding a shortest synchronizing word; a synchronizing set of maximal size (here also W[1]-hardness can be observed); or determining the rank of a given set. Further, by the observation (in [20]) that, in the construction given in [18, 11] the automata are WAAs, we immediately get NP-hardness for finding a shortest synchronizing word for all of our orders (for order  $l < l$  and  $l \leq l$  set  $R = \emptyset$ ).

## 5 Conclusion

We discussed ideas how constraints for the design of assembly lines caused by the physical deformation of a part can be described in terms of synchronization problems. For that, we considered several ways how a word can imply an order of states in  $Q$ . We considered the complexity of synchronizing an automaton under different variants of orders and observed that the complexity of considering an order on the set of active states may differ from considering the order on each single path. Although we were able to get a good understanding of the complexity of synchronization under the considered orders, some questions remained open: We only know that  $\text{SYNC-UNDER-}\alpha_{w@p}^{l<l}$  is contained in NP but it is open whether the problem is NP-complete or if it can be solved in polynomial time. Conversely, for  $\text{SYNC-UNDER-}\theta\text{-}\alpha_{w@p}^{l\leq l}$  the problem is NP-hard but its precise complexity is unknown. It would be quite surprising to observe membership in NP here since it would separate the complexity of this problem from the closely related problem  $\text{SYNC-UNDER-}1\text{-}\alpha_{w@p}^{l\leq l}$ . Further, it remains open whether for the other orders a drop in the complexity can be observed, when  $R$  is strict and total, as it is the case for  $\alpha_{w@p}^{l<f}$ .

---

## References

- 1 Journal of Automata, Languages and Combinatorics – Essays on the Černý Conjecture. [https://www.jalc.de/issues/2019/issue\\_24\\_2-4/content.html](https://www.jalc.de/issues/2019/issue_24_2-4/content.html). Accessed: 10/1/2020.
- 2 Dmitry S. Ananichev and Mikhail V. Volkov. Synchronizing monotonic automata. *Theor. Comput. Sci.*, 327(3):225–239, 2004.
- 3 Giorgio Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 1999.
- 4 Marie-Pierre Béal and Dominique Perrin. *Synchronised Automata*, page 213–240. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2016.
- 5 Jens Bruchertseifer and Henning Fernau. Synchronizing series-parallel automata with loops. In Rudolf Freund, Markus Holzer, and José M. Sempere, editors, *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019, Valencia, Spain, July 2-3, 2019.*, pages 63–78. Österreichische Computer Gesellschaft, 2019.
- 6 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovensk*, 14(3):208–215, 1964.
- 7 Ján Černý. A note on homogeneous experiments with finite automata. *Journal of Automata, Languages and Combinatorics*, 24(2-4):123–132, 2019.
- 8 Krishnendu Chatterjee and Laurent Doyen. Computation tree logic for synchronization properties. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 98:1–98:14, 2016.
- 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 10 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPICs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.
- 11 David Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510, 1990.
- 12 Henning Fernau, Vladimir V. Gusev, Stefan Hoffmann, Markus Holzer, Mikhail V. Volkov, and Petra Wolf. Computational complexity of synchronization under regular constraints. In Peter

- Rossmann, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 13 Henning Fernau, Pinar Heggernes, and Yngve Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *J. Comput. Syst. Sci.*, 81(4):747–765, 2015.
  - 14 Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybern.*, 14(1):105–115, 1999.
  - 15 Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory Comput. Syst.*, 54(2):293–304, 2014.
  - 16 Pavel V. Martyugin. Synchronization of automata with one undefined or ambiguous transition. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2012.
  - 17 Balas K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 132–142. IEEE Computer Society, 1986.
  - 18 I. K. Rystsov. On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of Ukrainian Acad. Sci., 1980. (in Russian).
  - 19 Igor K. Rystsov. Polynomial complete problems in automata theory. *Inf. Process. Lett.*, 16(3):147–151, 1983.
  - 20 Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. *Theor. Comput. Sci.*, 787:77–88, 2019.
  - 21 Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. *Fundam. Inform.*, 162(2-3):205–221, 2018.
  - 22 Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *Lecture Notes in Computer Science*, pages 5–33. Springer, 2004.
  - 23 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
  - 24 Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
  - 25 Avraham Trakhtman. The Černý conjecture for aperiodic automata. *Discrete Mathematics & Theoretical Computer Science*, 9(2), 2007.
  - 26 Uraz Cengiz Türker and Hüsnü Yenigün. Complexities of some problems related to synchronizing, non-synchronizing and monotonic automata. *Int. J. Found. Comput. Sci.*, 26(1):99–122, 2015.
  - 27 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008.
  - 28 Vojtech Vorel and Adam Roman. Parameterized complexity of synchronization and road coloring. *Discrete Mathematics & Theoretical Computer Science*, 17(1):283–306, 2015.
  - 29 Petra Wolf. Synchronization under dynamic constraints. *CoRR*, abs/1910.01935, 2019. [arXiv:1910.01935](https://arxiv.org/abs/1910.01935).