45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2025, December 17–19, 2025, BITS Pilani, K K Birla Goa Campus, India

Edited by

C. Aiswarya

Ruta Mehta

Subhajit Roy



LIPIcs – Vol. 360 – FSTTCS 2025 www.dagstuhl.de/lipics

Editors

C. Aiswarya
Chennai Mathematical Institute, India; IRL ReLaX, CNRS, France
aiswarya@cmi.ac.in

Ruta Mehta
University of Illinois at Urbana-Champaign, IL, USA
rutamehta@cs.illinois.edu

Subhajit Roy
Indian Institute of Technology Kanpur, India
subhajit@iitk.ac.in

ACM Classification 2012
Theory of computation; Computing methodologies; Software and its engineering

ISBN 978-3-95977-406-2

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/978-3-95977-406-2.

Publication date
December, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume 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 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/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.2025.0

ISBN 978-3-95977-406-2 **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

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
C. Aiswarya, Ruta Mehta, and Subhajit Roy 0:ix

Programme Committee
..... 0:xi

List of External Reviewers
..... 0:xiii

List of Authors
..... 0:xv

Invited Talks

Quantum Circuit Verification – A Potential Roadmap
Parosh Aziz Abdulla, Yu-Fang Chen, Michal Hečko, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, and Ramanathan Thinniyam Srinivasan 1:1–1:8

Unboundedness Problems for Formal Languages
Georg Zetzsche 2:1–2:10

Regular Papers

PDQMA = DQMA = NEXP: QMA with Hidden Variables and Non-Collapsing Measurements
Scott Aaronson, Sabee Grewal, Vishnu Iyer, Simon C. Marshall, and Ronak Ramachandran 3:1–3:17

How Pinball Wizards Simulate a Turing Machine
Rosemary U. Adejoh, Andreas Jakoby, Sneha Mohanty, and Christian Schindelhauer . 4:1–4:18

Distributed Games with a Central Decision Maker
Bharat Adsul and Nehul Jain 5:1–5:18

A Correct by Construction Fault Tolerant Voter for Input Selection of a Control System
Arif Ali AP, Jasine Babu, and Deepa Sara John 6:1–6:20

Quantum Protocols for Rabin Oblivious Transfer
Erika Andersson, Akshay Bansal, James T. Peat, Jamie Sikora, and Jiawei Wu .. 7:1–7:18

Parameterized Verification of Timed Networks with Clock Invariants
Étienne André, Swen Jacobs, Shyam Lal Karra, and Ocan Sankur 8:1–8:19

ε-Stationary Nash Equilibria in Multi-Player Stochastic Graph Games
Ali Asadi, Léonard Brice, Krishnendu Chatterjee, and K. S. Thejaswini 9:1–9:17

Cat Herding Game Played on Infinite Trees
Rylo Ashmore and Sophie Pinchinat 10:1–10:15

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Stability Notions for Hospital Residents with Sizes
Haricharan Balasundaram, J. B. Krishnashree , Girija Limaye, and Meghana Nasre 11:1–11:18

Explorability in Pushdown Automata
Ayaan Bedi and Karoliina Lehtinen 12:1–12:18

Synthesising Full-Information Protocols
Dietmar Berwanger, Laurent Doyen, and Thomas Soullard 13:1–13:18

Regulating Synchronous Data Exchange to Meet Control Flow and Data Specifications
Ashwin Bhaskar and M. Praveen 14:1–14:19

Extending EFX Allocations to Further Multi-Graph Classes
Umang Bhaskar and Yeshwant Pandit 15:1–15:18

The Agafonov and Schnorr-Stimm Theorems for Probabilistic Automata
Laurent Bienvenu, Hugo Gimbert, and Subin Pulari 16:1–16:15

Cutoff for the Swendsen–Wang Dynamics on the Complete Graph
Antonio Blanca and Zhezheng Song 17:1–17:21

Iterating Non-Aggregative Structure Compositions
Marius Bozga, Radu Iosif, and Florian Zuleger 18:1–18:18

Clustering in Varying Metrics
Deeparnab Chakrabarty, Jonathan Conroy, and Ankita Sarkar 19:1–19:21

Flavors of Quantifiers in Hyperlogics
Marek Chalupa, Thomas A. Henzinger, and Ana Oliveira da Costa 20:1–20:18

Overlay Network Construction: Improved Overall and Node-Wise Message Complexity
Yi-Jun Chang, Yanyu Chen, and Gopinath Mishra 21:1–21:21

IPS Lower Bounds for Formulas and Sum of ROABPs
Prerona Chatterjee, Utsab Ghosal, Partha Mukhopadhyay, and Amit Sinhababu ... 22:1–22:21

Parallel Complexity of Depth-First-Search and Maximal Path in Restricted Graph Classes
Archit Chauhan, Samir Datta, and M. Praveen 23:1–23:21

Languages of Words of Low Automatic Complexity Are Hard to Compute
Joey Chen, Bjørn Kjos-Hanssen, Ivan Koswara, Linus Richter, and Frank Stephan 24:1–24:19

On the Interplay of Cube Learning and Dependency Schemes in QCDCL Proof Systems
Abhimanyu Choudhury and Meena Mahajan 25:1–25:19

Characterizing NC^1 with Typed Monoids
Anuj Dawar and Aidan T. Evans 26:1–26:19

Fault-Tolerant Approximate Distance Oracles with a Source Set
Dipan Dey and Telikepalli Kavitha 27:1–27:15

Beyond Monads and Biproducts: A Uniform Interpretation of Parallelism in Intuitionistic Logic
Alejandro Díaz-Caro and Octavio Malherbe 28:1–28:17

Beyond Exact Fairness: Envy-Free Incomplete Connected Fair Division
Ajaykrishnan E S and Daniel Lokshtanov 29:1–29:16

Approximating Optimal Broadcast of Files in a Hose-Model Network
Thomas Erlebach, Naveen Garg, Sukriti Gupta, and Amitabh Trehan 30:1–30:19

Token Sliding Independent Set Reconfiguration on Block Graphs
Mathew C. Francis and Veena Prabhakaran 31:1–31:19

Fair Rent Division: New Budget and Rent Constraints
Rohith Reddy Gangam, Shayan Taherijam, and Vijay V. Vazirani 32:1–32:21

A Zone-Based Algorithm for Timed Parity Games
Gilles Geeraerts, Frédéric Herbreteau, Jean-François Raskin, and Alexis Reynouard 33:1–33:18

On the Hardness of Approximating Distances of Quantum Codes
Elena Grigorescu, Vatsal Jha, and Eric Samperton 34:1–34:18

Scalable Learning of One-Counter Automata via State-Merging Algorithms
Shibashis Guha, Anirban Majumdar, Prince Mathew, and A.V. Sreejith 35:1–35:19

Hardness of Finding Kings and Strong Kings
Ziad Ismaili Alaoui and Nikhil Mande 36:1–36:12

Communication Complexity of Equality and Error-Correcting Codes
Dale Jacobs, John Jeang, Vladimir Podolskii, Morgan Prior, and Ilya Volkovich .. 37:1–37:19

Fairness and Efficiency in Two-Sided Matching Markets
Pallavi Jain, Palash Jha, and Shubham Solanki 38:1–38:17

Improved Approximation for Pathwidth One Vertex Deletion and Parameterized Complexity of Its Variants
Satyabrata Jana, Soumen Mandal, Ashutosh Rai, and Saket Saurabh 39:1–39:20

Improved Upper Bounds on Multiflow-Multicut Gaps in Cactus Graphs
Sina Kalantarzadeh and Nikhil Kumar 40:1–40:19

Simultaneously Fair Allocation of Indivisible Items Across Multiple Dimensions
Yasushi Kawase, Bodhayan Roy, and Mohammad Azharuddin Sanpui 41:1–41:19

Degrees of Second and Higher-Order Polynomials
Donghyun Lim and Martin Ziegler 42:1–42:20

A Finer View of the Parameterized Landscape of Labeled Graph Contractions
Yashaswini Mathur and Prafullkumar Tale 43:1–43:19

Fall-Through Semantics for Mitigating Timing-Based Side Channel Leaks
Aniket Mishra and Abhishek Bichhawat 44:1–44:18

A Characterization of Spartan Graphs and New Lower Bounds for Eternal Vertex Cover
Neeldhara Misra and Saraswati Girish Nanoti 45:1–45:17

Finding Small Dijoins in Transitive Closure Time
Chaitanya Nalam and Thatchaphol Saranurak 46:1–46:11

The Algebraic Cost of a Boolean Sum
Ian Orzel, Srikanth Srinivasan, Sébastien Tavenas, and Amir Yehudayoff 47:1–47:13

On the Roots of Independence Polynomial: Quantifying the Gap
Om Prakash and Vikram Sharma 48:1–48:18

On the Hardness of Order Finding and Equivalence Testing for ROABPs
C. Ramya and Pratik Shastri 49:1–49:13

Additive, Near-Additive, and Multiplicative Approximations for APSP in Weighted Undirected Graphs: Trade-Offs and Algorithms
Liam Roditty and Ariel Sapir 50:1–50:22

Randomized Black-Box PIT for Small Depth +-Regular Non-Commutative Circuits
G. V. Sumukha Bharadwaj and S. Raja 51:1–51:16

Unreliability in Practical Subclasses of Communicating Systems
Amrita Suresh and Nobuko Yoshida 52:1–52:22

Preface

This volume contains the proceedings of the 45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025). The conference was held as an in-person event during the period December 17–19, 2025, on the campus of BITS Pilani - K K Birla Goa Campus, in Goa, India.

The conference has two tracks: Track A focusing on algorithms, complexity, and related optimization questions, and Track B focusing on logic, automata and other formal method aspects of computer science. Each track had its own Program Committee (PC) and chairs (Ruta Mehta for Track A and C. Aiswarya and Subhajit Roy for Track B). This volume constitutes the joint proceedings of the two tracks, published in the LIPIcs series under a Creative Common license, with free online access for all.

The conference comprises of 6 invited talks, 29 contributed talks in Track A, and 21 contributed talks in Track B. This volume contains all the contributed papers from the two tracks, and two articles accompanying two of the invited talks. We thank all the authors who submitted their papers to FSTTCS 2025. We are especially grateful to the PC members for their tireless work, and all the external reviewers for their expert opinion in the form of timely reviews.

We thank all the invited speakers for accepting our invitation: Parosh Aziz Abdulla (Uppsala University & Mälardalen University), Bernd Finkbeiner (CISPA), Nicole Immorlica (Yale University & Microsoft Research New England), Pravesh K Kothari (Princeton University), Barna Saha (University of California San Diego), and Georg Zetzsche (MPI-SWS, Kaiserslautern).

The main conference was co-located with eight workshops: Formal Verification and Learning for Quantitative Systems (organised by S. Akshay, Shibashis Guha, and Suguman Bansal), Workshop on Quantum Algorithms and Cryptography (organised by Akshay Bansal, Rajendra Kumar, and Rajat Mittal), Information in Games and Mechanisms (organised by Umang Bhaskar and Rohit Vaish), Workshop on Games and Automata for Synthesis (organised by Sougata Bose, Suman Sadhukhan, Thejaswini K. S.), 10th Indian SAT+SMT School (organised by Supratik Chakraborty, Priyanka Golia, Ashutosh Gupta, Saurabh Joshi, Kumar Madhukar, Kuldeep Meel, and Aalok Thakkar), Mentoring Workshop for senior undergrad and early PhD students (organised by Neeldhara Misra, KC Sivaramakrishnan, and Aalok Thakkar), Compilers-oriented Software Security (organised by Awanish Pandey and Raghvendra Singh Rohit), and Research Highlights in Programming Languages (organised by Abhisekh Sankaran, Divyesh Unadkat, Deepak D'Souza, Uday Khedker, Kumar Madhukar, Kartik Nagar, Sumanth Prabhu, Ganesan Ramalingam, and Abhik Roychoudhury).

We are indebted to the organizing committee from BITS Pilani, K K Birla Goa Campus, namely, Yasmeen Akhtar, Siddharth Gupta (chair), Mukesh Kumar Nagar, Gargi Sanket Prabhu, Hemant Rathore, Aniket Basu Roy, Tarkeshwar Singh, and Subhadip Singha, as well as to A. Baskar (BITS Pilani, K K Birla Goa Campus) for making all the necessary arrangements for the conference. We are thankful to the past organizing chair, Neeldhara Misra (IIT-Gandhinagar), for knowledge transfer.

We thank S.P. Suresh (Chennai Mathematical Institute) for maintaining the conference web page as well as the publicity chairs Neeldhara Misra (IIT Gandhinagar), Shibashis Guha (TIFR Mumbai), and Aalok Thakkar (Asoka University, Sonepat). We are grateful to the friendly staff at Dagstuhl LIPIcs for helping us put together the proceedings, in

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

particular to Michael Didas and Michael Wagner. Finally, we thank the members of the Steering Committee, especially Deepak D'Souza (IISc), and the PC Chairs from FSTTCS 2024 (Siddharth Barman (IISc) and Sławomir Lasota (University of Warsaw)), for providing pertinent information and valuable advice about various aspects of the conference.

C. Aiswarya, Ruta Mehta, and Subhajit Roy
October 2025

Programme Committee

Track A

- Divesh Aggarwal (National University of Singapore)
- Arpit Agrawal (Indian Institute of Technology Bombay)
- John Augustine (Indian Institute of Technology Madras)
- Siddharth Barman (Indian Institute of Science)
- Greg Bodwin (University of Michigan)
- Ashish Chiplunkar (Indian Institute of Technology Delhi)
- Sayak Ray Chowdhury (Indian Institute of Technology Kanpur)
- Alex Hollender (University of Oxford)
- Dakshita Khurana (University of Illinois Urbana-Champaign)
- Pravesh Kothari (Princeton University)
- Vasilis Livanos (EPFL, Lausanne)
- Nikhil Mande (University of Liverpool)
- Kurt Mehlhorn (Max Planck Institute for Informatics)
- Ruta Mehta (University of Illinois at Urbana-Champaign) – Track A chair
- Divyarthi Mohan (Boston University)
- Prajakta Nimbhorkar (Chennai Mathematical Institute)
- Ioannis Panageas (University of California, Irvine)
- Ramprasad Saptharishi (Tata Institute of Fundamental Research Mumbai)
- Saket Saurabh (The Institute of Mathematical Sciences)
- Laci Vegh (University of Bonn)
- Jiaheng Wang (University of Regensburg)

Track B

- C Aiswarya (Chennai Mathematical Institute) – Track B co-chair
- Ahmed Bouajjani (IRIF, Université Paris Cité)
- Julie Cailler (University of Lorraine)
- Supratik Chakraborty (Indian Institute of Technology Bombay)
- Krishnendu Chatterjee (Institute of Science and Technology Austria)
- Lorenzo Clemente (University of Warsaw)
- Ugo Dal Lago (University of Bologna)
- Stéphane Demri (LMF, CNRS & ENS Paris-Saclay)
- Manfred Droste (Leipzig University)
- Stefan Göller (University of Kassel)
- Giulio Guerrieri (university of Sussex)
- Ashutosh Gupta (Indian Institute of Technology Bombay)
- Sandra Kiefer (University of Oxford)
- Dietrich Kuske (Technische Universität Ilmenau)
- Roman Kuznets (Czech Academy of Sciences)
- Benjamin Monmege (Aix-Marseille University)
- Guillermo A. Pérez (University of Antwerp)
- Sophie Pinchinat (University of Rennes)
- Subhajit Roy (Indian Institute of Technology Kanpur) – Track B co-chair
- Prakash Saivasan (The Institute of Mathematical Sciences)

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Ocan Sankur (Mitsubishi Electric R&D Centre Europe)
- Alexis Saurin (IRIF, CNRS)
- Sunil Simon (Indian Institute of Technology Kanpur)
- A V Sreejith (Indian Institute of Technology Goa)
- S P Suresh (Chennai Mathematical Institute)

List of External Reviewers

Konrad Anand
Eugene Asarin
Nikhil Balaji
Akshay Bansal
Niel De Beaudrap
Dylan Bellier
Vishwas Bhargava
Markus Bläser
Cornelius Brand
Debarshi Chanda
Abhranil Chatterjee
Prerona Chatterjee
Rohan Chauhan
Mitchell Chiew
Suryajith Chillara
Thomas Colcombet
Arjan Cornelissen
Antonio Cruciani
Mary Cryan
Peter Csikvari
Gergely Csáji
Nirjhar Das
Davide Davoli
Devdan Dey
Simon Döring
Baris Can Esmer
Stephan Felber
Weiming Feng
Mathew Francis
Graham Freifeld
Andreas Galanis
Hugo Rincon Galeana
Naveen Garg
Sutanu Gayen
Utsab Ghosal
Sarbajit Ghosh
Matan Gilboa
Hugo Gimbert
Frederik Glitzner
Tiger-Lily Goldsmith
Svenja M. Griesbach
Shibashis Guha
Nikhil Gupta
Rishav Gupta
Shivam Gupta
Ryan Hota
Farzad Jafarrahmani
Satyabrata Jana
William K. Moses Jr.
Sotiris Kanellopoulos
Bruce Kapron
Upendra Kapshikar
Debajyoti Kar
Matthias Kaul
Max Klimm
Zhuan Khye Koh
Andreas Kontogiannis
Rucha Kulkarni
Manish Kumar
Rajendra Kumar
Srijita Kundu
Wenzheng Li
Zeyong Li
Siyue Liu
Felipe Garrido Lucero
Manideep Mamindlapally
Giulio Manzonetto
Simon Mauras
Nikolaos Melissinos
Mirabel Mendoza-Cadena
Aditya Morolia
Ananta Mukherjee
Anish Mukherjee
Saswata Mukherjee
Kartik Nagar
Satyadev Nandakumar
Vishnu Narayan
Virendra Nishad
Jakob Nogler
Shreyas Pai
Akash Pareek
Prathamesh Patil
Nikolas Patris
Subhasree Patro
Chris Pergaminelis
Geevarghese Philip
Elaine Pimentel
Vasilis Pollatos
Tomasz Ponitka
Tephilla Prince

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Romain Péchoux
Nithish Raja
Srikkanth Ramachandran
R Ramanujam
C Ramya
Nidhi Rathi
Christian Rauch
Ahmed Rezine
Kilian Risse
Simon Robillard
Jonathan Rollin
Kilian Rothmund
Sasanka Roy
Janosch Ruff
Prasanth Kumar A S
Irmak Saglam
Chandan Saha
Govind S. Sankar
Uddalok Sarkar
Jayalal Sarma
Ignasi Sau
Roopam Saxena
Šimon Schierreich
Daniel Schoepflin
Jens Kristian Refsgaard Schou
Aditi Sethia
Parnian Shahkar
Shuai Shao
Pratik Shastri
Sudarshan Shyam
Makrand Sinha
Taylor Smith
Karnati Venkata Naga Sreenivasulu
Xizhi Tan
Anamay Tengse
Sharma Thankachan
Mukesh Tiwari
Paritosh Verma
Gautham Vishwanathan
Ilya Volkovich
Jingming Yan
Mingyang Yang
Alice Zheng
Martin Zimmermann

List of Authors

Scott Aaronson (3)
University of Texas at Austin, TX, USA

Parosh Aziz Abdulla (1)
Uppsala University, Sweden; Mälardalen University, Sweden

Rosemary U. Adejoh (4)
Faculty of Media, Bauhaus-Universität Weimar, Germany

Bharat Adsul (5)
Indian Institute of Technology Bombay, India

Arif Ali AP (6)
Department of Computer Science and Engineering, Indian Institute of Technology Palakkad, India

Erika Andersson (7)
SUPA, Institute of Photonics and Quantum Sciences, School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, UK

Étienne André (8)
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France; Institut Universitaire de France (IUF), Paris, France

Ali Asadi (9)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Rylo Ashmore (10)
Memorial University of Newfoundland, St. John's, Canada

Jasine Babu (6)
Department of Computer Science and Engineering, Indian Institute of Technology Palakkad, India

Haricharan Balasundaram (11)
Indian Institute of Technology Madras, Chennai, India

Akshay Bansal (7)
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Ayaan Bedi (12)
Chennai Mathematical Institute, India

Dietmar Berwanger (13)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

Ashwin Bhaskar (14)
Chennai Mathematical Institute, India

Umang Bhaskar (15)
Tata Institute of Fundamental Research, Mumbai, India

Abhishek Bichhawat (44)
Indian Institute of Technology Gandhinagar, India

Laurent Bienvenu (16)
LaBRI, CNRS & Université de Bordeaux, Talence, France

Antonio Blanca (17)
Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA

Marius Bozga (18)
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Léonard Brice (9)
Université Libre de Bruxelles, Belgium

Deeparnab Chakrabarty (19)
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Marek Chalupa (20)
IST Austria, Klosterneuburg, Austria

Yi-Jun Chang (21)
National University of Singapore, Singapore

Krishnendu Chatterjee (9)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Prerona Chatterjee (22)
School of Computer Sciences, NISER, Bhubaneswar, India

Archit Chauhan (23)
Department of Computer Science and Engineering, IIT Bombay, India

Joey Chen (24)
Department of Mathematics, National University of Singapore, Singapore

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Yanyu Chen (21)
National University of Singapore, Singapore

Yu-Fang Chen (1)
Academia Sinica, Taipei, Taiwan

Abhimanyu Choudhury (25)
The Institute of Mathematical Sciences, Chennai, India; Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Jonathan Conroy (19)
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Samir Datta (23)
Chennai Mathematical Institute, India; UMI ReLaX, Indo-French joint research unit, Chennai, India

Anuj Dawar (26)
Department of Computer Science and Technology, University of Cambridge, UK

Dipan Dey (27)
Tata Institute of Fundamental Research, Mumbai, India

Laurent Doyen (13)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

Alejandro Díaz-Caro (28)
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France; Universidad Nacional de Quilmes, Buenos Aires, Argentina

Ajaykrishnan E S (29)
University of California, Santa Barbara, CA, USA

Thomas Erlebach (30)
Department of Computer Science, Durham University, UK

Aidan T. Evans (26)
Department of Computer Science and Technology, University of Cambridge, UK

Mathew C. Francis (31)
Indian Institute of Technology, Palakkad, India

Rohith Reddy Gangam (32)
University of California, Irvine, CA, USA

Naveen Garg (30)
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Gilles Geeraerts (33)
Université libre de Bruxelles, Belgium

Utsab Ghosal (22)
Chennai Mathematical Institute, India

Hugo Gimbert (16)
LaBRI, CNRS & Université de Bordeaux, Talence, France

Sabee Grewal (3)
University of Texas at Austin, TX, USA

Elena Grigorescu (34)
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Shibashis Guha (35)
Tata Institute of Fundamental Research, Mumbai, India

Sukriti Gupta (30)
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Thomas A. Henzinger (20)
IST Austria, Klosterneuburg, Austria

Frédéric Herbreteau (33)
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 33400 Talence, France

Michal Hečko (1)
Brno University of Technology, Czech Republic

Lukáš Holík (1)
Brno University of Technology, Czech Republic

Radu Iosif (18)
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Ziad Ismaili Alaoui (36)
Department of Computer Science, University of Liverpool, UK

Vishnu Iyer (3)
University of Texas at Austin, TX, USA

Dale Jacobs (37)
Tufts University, Medford, MA, USA

Swen Jacobs (8)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Nehul Jain (5)
Indian Institute of Technology Bombay, India

Pallavi Jain (38)
Indian Institute of Technology Jodhpur, India

Andreas Jakoby (4)
Faculty of Media, Bauhaus-Universität Weimar, Germany

Satyabrata Jana (39)
University of Warwick, UK

John Jeang (37)
Tufts University, Medford, MA, USA

Palash Jha (38)
Indian Institute of Technology Jodhpur, India

Vatsal Jha (34)
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Deepa Sara John (6)
ISRO Inertial Systems Unit, Indian Space Research Organization, Thiruvananthapuram, India

Sina Kalantarzadeh (40)
University of Waterloo, Waterloo, Canada

Shyam Lal Karra (8)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Telikepalli Kavitha (27)
Tata Institute of Fundamental Research, Mumbai, India

Yasushi Kawase (41)
The University of Tokyo, Japan

Bjørn Kjos-Hanssen (24)
Department of Mathematics, University of Hawai'i at Mānoa, Honolulu, HI, USA

Ivan Koswara (24)
School of Computing, National University of Singapore, Singapore

J. B. Krishnashree (11)
Tata Institute of Fundamental Research Mumbai, India

Nikhil Kumar (40)
Tata Institute of Fundamental Research, Mumbai, India

Karoliina Lehtinen (12)
CNRS Researcher, Aix-Marseille Université, LIS, France

Ondřej Lengál (1)
Brno University of Technology, Czech Republic

Donghyun Lim (42)
KAIST, Daejeon, Republic of Korea

Girija Limaye (11)
Birla Institute of Technology and Science, Pilani, K K Birla Goa Campus, India

Jyun-Ao Lin (1)
National Taipei University of Technology, Taiwan

Daniel Lokshtanov (29)
University of California, Santa Barbara, CA, USA

Meena Mahajan (25)
The Institute of Mathematical Sciences, Chennai, India; Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Anirban Majumdar (35)
Independent researcher, Kolkata, India

Octavio Malherbe (28)
IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

Soumen Mandal (39)
Department of Mathematics, IIT Delhi, India

Nikhil Mande (36)
Department of Computer Science, University of Liverpool, UK

Simon C. Marshall (3)
Leiden University, The Netherlands

Prince Mathew (35)
Indian Institute of Technology Goa, Ponda, India

Yashaswini Mathur (43)
Indian Institute of Science Education and Research Bhopal, India

Aniket Mishra (44)
Indian Institute of Technology Gandhinagar, India

Gopinath Mishra (21)
National University of Singapore, Singapore

Neeldhara Misra (45)
IIT Gandhinagar, India

Sneha Mohanty (4)
Computer Networks and Telematics, University of Freiburg, Germany

Partha Mukhopadhyay (22)
Chennai Mathematical Institute, India

Chaitanya Nalam (46)
University of Michigan, Ann Arbor, MI, USA

Saraswati Girish Nanoti (45)
Indian Institute of Science, Bangalore, India

Meghana Nasre (11)
Indian Institute of Technology Madras, Chennai, India

Ana Oliveira da Costa (20)
IST Austria, Klosterneuburg, Austria

Ian Orzel (47)
Department of Computer Science, The University of Copenhagen, Denmark

Yeshwant Pandit (15)
Tata Institute of Fundamental Research, Mumbai, India

James T. Peat (7)
SUPA, Institute of Photonics and Quantum Sciences, School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, UK

Sophie Pinchinat (10)
IRISA/University of Rennes, France

Vladimir Podolskii (37)
Tufts University, Medford, MA, USA

Veena Prabhakaran (31)
IITM Pravartak Technologies Foundation, Chennai, India

Om Prakash (48)
The Institute of Mathematical Sciences, HBNI, Chennai, India

M. Praveen (14, 23)
Chennai Mathematical Institute, India; CNRS IRL ReLaX, Chennai, India

Morgan Prior (37)
Tufts University, Medford, MA, USA

Subin Pulari (16)
LaBRI, CNRS & Université de Bordeaux, Talence, France

Ashutosh Rai (39)
Department of Mathematics, IIT Delhi, India

S. Raja (51)
Department of Computer Science & Engineering, Indian Institute of Technology Tirupati, India

Ronak Ramachandran (3)
University of Texas at Austin, TX, USA

C. Ramya (49)
The Institute of Mathematical Sciences, Chennai, India

Jean-François Raskin (33)
Université libre de Bruxelles, Belgium

Alexis Reynouard (33)
Université libre de Bruxelles, Belgium

Linus Richter (24)
Department of Mathematics, National University of Singapore, Singapore

Liam Roditty (50)
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Bodhayan Roy (41)
Indian Institute of Technology Kharagpur, India

Eric Samperton (34)
Departments of Mathematics and Computer Science, Purdue Quantum Science and Engineering Institute, Purdue University, West Lafayette, IN, USA

Ocan Sankur (8)
Université de Rennes, CNRS, Inria, Rennes, France

Mohammad Azharuddin Sanpui (41)
Indian Institute of Technology Kharagpur, India

Ariel Sapir (50)
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Thatchaphol Saranurak (46)
University of Michigan, Ann Arbor, MI, USA

Ankita Sarkar (19)
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Saket Saurabh (39)
The Institute of Mathematical Sciences, HBNI, Chennai, India; University of Bergen, Norway

Christian Schindelhauer (4)
Computer Networks and Telematics, University of Freiburg, Germany

Vikram Sharma (48)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Pratik Shastri (49)
The Institute of Mathematical Sciences, Chennai, India

Jamie Sikora (7)
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Amit Sinhababu (22)
Chennai Mathematical Institute, India

Shubham Solanki (38)
Indian Institute of Technology Jodhpur, India

Zhezheng Song (17)
Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA

Thomas Soullard (13)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

A.V. Sreejith (35)
Indian Institute of Technology Goa, Ponda, India

Ramanathan Thinniyam Srinivasan (1)
Uppsala University, Sweden

Srikanth Srinivasan (47)
Department of Computer Science, The University of Copenhagen, Denmark

Frank Stephan (24)
Department of Mathematics, National University of Singapore, Singapore; School of Computing, National University of Singapore, Singapore

G. V. Sumukha Bharadwaj (51)
Department of Computer Science & Engineering, Indian Institute of Technology Tirupati, India

Amrita Suresh (52)
University of Oxford, UK

Shayan Taherijam (32)
University of California, Irvine, CA, USA

Prafullkumar Tale (43)
Indian Institute of Science Education and Research Pune, India

Sébastien Tavenas (47)
Chargé de Recherche CNRS au Laboratoire LAMA de l'Université Savoie Mont Blanc, France

K. S. Thejaswini (9)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Amitabh Trehan (30)
Department of Computer Science, Durham University, UK

Vijay V. Vazirani (32)
University of California, Irvine, CA, USA

Ilya Volkovich (37)
Boston College, MA, USA

Jiawei Wu (7)
Centre for Quantum Technologies, National University of Singapore, Singapore

Amir Yehudayoff (47)
Department of Computer Science, The University of Copenhagen, Denmark; Department of Mathematics, Technion-IIT, Haifa, Israel

Nobuko Yoshida (52)
University of Oxford, UK

Georg Zetzsche (2)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Martin Ziegler (42)
KAIST, Daejeon, Republic of Korea

Florian Zuleger (18)
Institute of Logic and Computation, Technische Universität Wien, Austria

Quantum Circuit Verification – A Potential Roadmap

Parosh Aziz Abdulla
Uppsala University, Sweden
Mälardalen University, Sweden

Yu-Fang Chen
Academia Sinica, Taipei, Taiwan

Michal Hečko
Brno University of Technology, Czech Republic

Lukáš Holík
Brno University of Technology, Czech Republic

Ondřej Lengál
Brno University of Technology, Czech Republic

Jyun-Ao Lin
National Taipei University of Technology, Taiwan

Ramanathan Thinniyam Srinivasan
Uppsala University, Sweden

Abstract

Quantum technologies are progressing at an extraordinary pace and are poised to transform numerous sectors both nationally and globally. Among them, quantum computing stands out for its potential to revolutionize areas such as cryptography, optimization, and the simulation of quantum systems, offering dramatic speed-ups for specific classes of problems.

As quantum devices evolve and become increasingly pervasive, guaranteeing their correctness is of paramount importance. This necessitates the development of rigorous methods and tools to analyze and verify their behavior. However, the construction of such verification frameworks presents fundamental challenges. Quantum phenomena such as superposition and entanglement give rise to computational behaviors that differ profoundly from those of classical systems, leading to inherently probabilistic models and exponentially large state spaces, even for relatively small programs.

Addressing these challenges requires building on the extensive expertise of the formal methods community in classical program verification, while incorporating recent advances and collaborative efforts in quantum systems. An interesting challenge for the verification community is to design and implement novel verification frameworks that transfer the key strengths of classical verification, such as expressive specification, precise error detection, automation, and scalability, to the quantum domain. We expect that the results of this research will play a crucial role in enabling the dependable deployment of quantum technologies across a wide range of future applications.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Quantum Circuits, Quantum Computing, Program Verification, Automata, Model Checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.1

Category Invited Talk



© Parosh Aziz Abdulla, Yu-Fang Chen, Michal Hečko, Lukáš Holík, Ondřej Lengál, Jyun-Ao Lin, and Ramanathan Thinniyam Srinivasan;
licensed under Creative Commons License CC-BY 4.0

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).
Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy; Article No. 1; pp. 1:1–1:8



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

000	0	100	0
001	0	101	$\frac{1}{3}$
010	0	110	$\frac{1}{3}$
011	$\frac{\sqrt{6}}{3}$	111	$\frac{1}{3}$

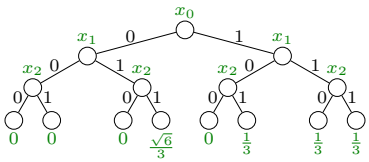


Figure 1 A quantum state with three qubits and its tree representation. The system can be in any of the eight basis states, with the corresponding probability amplitudes indicated in the table.

1 Background

Classical computing has undergone remarkable advances over the past decades and today forms the backbone of virtually all modern technological infrastructure. By contrast, *quantum computing* introduces a fundamentally different paradigm for information processing, one that is deeply rooted in the principles of quantum mechanics. This new model holds the potential to solve computational problems that are intractable by classical means.

The essential difference between classical and quantum computation lies in how information is represented and manipulated. Classical computers use *bits*, which can exist only in one of two binary states, 0 or 1. Quantum computers, on the other hand, operate on *quantum bits* (*qubits*), which obey the principles of *superposition* and *entanglement*. Superposition allows a qubit to exist in a combination of states simultaneously, enabling parallel exploration of many computational possibilities. Entanglement introduces correlations between qubits such that the state of one qubit can instantaneously influence the state of another, regardless of physical separation. Together, these features enable the simultaneous representation and manipulation of an exponentially large number of states and allow the encoding of complex correlations that classical systems cannot efficiently replicate.

Figure 1 illustrates a quantum state composed of three qubits.[1] In a classical setting, the system would be restricted to a single *basis state* among the eight possible configurations $000, 001, \ldots, 111$. In contrast, a quantum system may exist in a superposition of all eight basis states simultaneously, each associated with a given *probability amplitude*.

A quantum state can therefore be viewed as a distribution over basis states, where each state is assigned a complex amplitude. This distribution can be represented as a tree: each path from the root to a leaf corresponds to a basis state, and each leaf stores the associated amplitude. Unlike classical probabilistic systems, where probabilities are real and sum to one, quantum amplitudes are complex numbers whose squared absolute values sum to one.[2]

A major milestone in the field is the achievement of *quantum supremacy*, the point at which a quantum computer can solve a problem beyond the reach of any classical supercomputer. Achieving quantum supremacy would yield exponential performance improvements on certain computational tasks. Although practical quantum computing remains in its early stages, applications are advancing rapidly. In the near term, *hybrid systems* that integrate classical and quantum processors are expected to become increasingly common. In the longer term, quantum supremacy has the potential to transform entire industries by enabling solutions to previously intractable problems.

[1] Throughout this document, we use simplified examples to highlight the verification challenges arising in quantum circuits. Detailed technical presentations can be found, for example, in [20].

[2] An *amplitude* generalizes the notion of probability. The squared magnitude of a complex amplitude corresponds to the probability of observing a basis state. The use of complex numbers allows for interference effects, where contributions with opposite phases cancel each other, enabling phenomena such as "negative probabilities" after squaring.

Promising application domains for quantum technologies include medical diagnostics, cryptography, secure communication infrastructures, and autonomous systems, among others. These areas require extremely high reliability and cannot tolerate critical system failures, making rigorous certification and assurance processes indispensable. Ensuring the dependability of quantum systems is therefore a central requirement for their widespread adoption.

However, verifying quantum systems is inherently challenging. Their probabilistic nature, combined with the exponential growth of state spaces as the number of qubits increases, creates profound obstacles for formal reasoning. Addressing these challenges demands new theoretical foundations and algorithmic techniques.

In our research, we are currently focusing on the *formal verification of quantum circuits*. Our approach is twofold: (i) to develop precise computational models that faithfully represent quantum state spaces and (ii) to design algorithmic techniques for verifying the behavior of quantum circuits [10]. In parallel with these theoretical efforts, we are building verification tools and evaluate them extensively through experiments on a diverse set of applications and case studies.

2 Purpose and Goals

It is essential to develop methods, algorithms, and tools for the automated verification of quantum systems, with particular focus on the unique challenges arising from their fundamentally different behavior. We envision significant added value in bridging two complementary areas of computer science, especially when established techniques from a mature field are adapted to address complex problems in an emerging domain. Such work embodies such interdisciplinary integration by applying well-developed methods from logic, automata theory, and symbolic verification to the problem of ensuring the correctness of quantum programs.

A particularly promising direction is the adaptation of techniques from automata theory, a foundational discipline in formal verification, to the verification of quantum systems. It would be interesting to use automata-based reasoning with symbolic representations tailored for the automated analysis of quantum behavior. The goal is to generalize core concepts from classical verification, such as state-space exploration and symbolic reasoning, into the quantum realm, thereby enabling the adaptation of robust classical paradigms to quantum computing, where formal assurances are essential.

We believe that such an approach also opens new avenues for research in automata theory itself. The mathematical structures encountered in quantum systems introduce both novel challenges and opportunities, extending the expressive and analytical power of automata-based techniques and fostering new theoretical developments. It establishes a conceptual bridge between quantum program verification and automata theory, fostering potential for new theoretical developments and extending the expressive and analytical power of automata-based techniques into the context of quantum computation. To achieve these goals, we have identified four specific objectives.

- We need to develop *symbolic representations*, based on *automata*, that serve as a theoretical and algorithmic foundation for efficiently modeling the state spaces of quantum circuits. These representations are grounded in the idea of modeling quantum states as binary trees, where each path from the root to a leaf corresponds to a computational basis state (see Figure 1). We would like to design new algorithms and establish complexity results for fundamental operations – such as language inclusion, successor computation, minimization, (bi)simulation, as well as predecessor and successor state analysis – which are vital for performing symbolic verification of quantum systems.

- We need to establish a framework for *algorithmic verification* of quantum systems. This framework will be based on a regular model checking approach, employing automata as symbolic representations of quantum state spaces. A central challenge in this effort is parameterized verification, where the goal is to prove correctness regardless of the number of input qubits or computational steps. This requires the development of new techniques for reasoning about infinite families of quantum states and operations, which is a significant departure from traditional finite-state verification methods.
- It is important to identify *key areas* of quantum computing where verification is critically needed. These include quantum measurement and circuit equivalence checking. By instantiating our methods in these application areas, we can try to demonstrate both the generality and the practical value of our verification techniques.
- To increase impact, we need to undertake a focused *implementation effort*. Our own existing tool, AutoQ [9, 10], is a case in point. It serves as a basis for implementing the above mentioned algorithms. We need to evaluate such tools extensively using a variety of representative applications and case studies, thereby validating the effectiveness and scalability of our approach.

We believe these objectives will help make significant contributions to the field of quantum computing by providing robust methods for ensuring the correctness and reliability of quantum systems.

3 Current Limitations and Potential Solutions

Research on the algorithmic verification of quantum systems is still in its infancy. Only recently have the first foundational contributions appeared, including initial work employing SAT solvers for quantum verification [15]. A number of recent studies have further advanced the field by introducing approaches for symbolic verification of quantum circuits [10, 2, 9].

By contrast, the algorithmic verification of classical systems, particularly through *model checking* [12], has been a mature and active research area for nearly four decades. This line of work has yielded a substantial and well-established body of theory and practice, and continues to evolve rapidly, with results regularly published in leading venues such as POPL, CAV, PLDI, MICRO, and ASPLOS.

While extending verification techniques to quantum systems is a natural and necessary progression, it introduces a range of new and technically demanding challenges, as discussed in Section 1 and Section 2. The following sections review the current state of the art, outline its limitations, and summarize key results in both quantum and classical program verification that inform future research directions.

3.1 Symbolic Encodings

Symbolic state-space representations have been among the most powerful techniques in classical program verification over the past three decades. By representing program semantics symbolically, using logical formulas and decision structures, verification tools can efficiently reason about large or infinite state spaces and check correctness properties. Binary Decision Diagrams (BDDs) have been instrumental in hardware and protocol verification, while automata over words and trees have enabled the verification of parameterized systems and programs with dynamic data structures.

Despite their success in classical settings, symbolic techniques have so far seen limited application in quantum verification. The mathematical foundations of quantum computation – superposition, probabilistic measurement, and non-local entanglement – pose fundamental

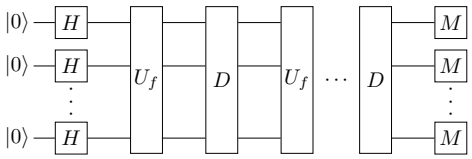


Figure 2 The classical Grover's search algorithm.

obstacles for classical symbolic reasoning. Nevertheless, suitably adapted symbolic methods are expected to play a crucial role in the scalability and precision of quantum program verification.

Recent work has introduced symbolic representations for sets of quantum states based on extended classes of tree automata [10, 9]. These automata generalize classical tree automata by allowing transitions labeled with sets of choices and enabling synchronization between subtrees within an accepted tree. Importantly, they retain key theoretical properties such as closure under union and intersection, and decidability of language emptiness and inclusion. Building on this foundation, new fully automated symbolic verification algorithms for quantum circuits have been proposed, supporting quantum gate operations with at most quadratic complexity – an improvement over earlier tree-automata-based methods with exponential worst-case complexity.

In the classical setting, numerous symbolic encodings have been proposed for infinite-state systems, including those based on automata, well-quasi-ordered constraint systems, and linear arithmetic. These techniques have been successfully applied to communication protocols, distributed algorithms, and real-time systems. Advances in automata-based verification include new simulation relations for tree automata and efficient algorithms for their computation [5], as well as the first notions of bisimulation for tree automata [11, 6], where the classical Paige–Tarjan algorithm was extended from word automata to tree automata. Subsequent work introduced additional bisimulation relations whose computation reduces to checking bisimulation over word automata [7].

Despite these advances, several limitations remain in the context of quantum verification.

- With the exception of [14] and a handful of more recent studies [10, 9], there is currently little work on symbolic encodings specifically tailored to the automated verification of quantum circuits. Existing approaches, such as quantum Hoare logic [17, 21], define predicates as mappings from mixed states to real values in $[0, 1]$, representing the probability that a state satisfies a given condition. These logics are inherently deductive and interactive, requiring significant manual reasoning, and are therefore unsuitable for automated verification.

  A fundamentally different direction is a *set-based* approach, where predicates map quantum states to $\{0, 1\}$. Automata can then serve as compact representations of such predicates: a tree representing a quantum state is accepted if and only if the predicate evaluates to 1. This formulation facilitates the construction of efficient and fully automated verification algorithms, offering much greater scalability in practical applications.

- No algorithms currently exist for minimization, simulation, or bisimulation over automata models specifically designed for quantum verification. Developing such algorithms remains a central open problem and requires adapting classical automata-theoretic techniques to account for the unique algebraic structure and non-local behavior of quantum systems.

3.2 Algorithmic Verification

Algorithmic verification techniques, most prominently *model checking*, have been among the most influential approaches for verifying classical systems over the past three decades. They provide fully automated methods for determining whether a system satisfies a given specification and, unlike testing, can guarantee the absence of errors. Extending such techniques to quantum systems introduces fundamental challenges.

First, quantum systems naturally involve infinite state spaces, even when composed of a finite number of qubits. Second, they are inherently parameterized, often in multiple dimensions. For example, the circuit shown in Figure 2 implements Grover's search algorithm, which operates on an n-bit Boolean function and achieves a quadratic speed-up over classical search, requiring $O(\sqrt{N})$ iterations for $N = 2^n$. Both the number of input qubits tot he circuit and the number of computational stages (the number of gates) are unbounded, and correctness must hold across this entire parameter space.

Third, quantum systems exhibit *globally entangled transitions*: applying a gate to a single qubit can affect an exponential number of classical basis states. For instance, in Figure 1, negating qubit x_1 swaps multiple leaf positions simultaneously, demonstrating the non-local nature of quantum transformations.

Recent work [10, 9] has introduced verification tools for quantum circuits that are fully automated, support expressive property specification, and generate informative bug traces. These tools use a variant of tree automata [16] to represent sets of quantum states and to perform symbolic execution of quantum gates. Their improved scalability compared to traditional approaches is largely due to *level synchronization*, which constrains quantum operations to qubits located at the same tree level (Section 1).

In the classical domain, significant progress has been achieved in the verification of infinite-state programs, including systems operating under weak memory models [4, 3, 8]. Parameterized verification has been extensively studied and provides methods for proving correctness independently of the number of system components [12]. One particularly successful paradigm is *Regular Model Checking* [1], which represents sets of states using automata and transition relations using transducers. Reachability is determined by iteratively applying the transducer to the initial states and checking for intersection with the set of target states.

In a similar manner to above, there are several limitations in the current state of the art for quantum verification.

- Approaches such as SYMQV [13] use symbolic execution [18] and SMT solving to verify input-output relationships but face scalability limitations. The SMT-based approach of [15] improves encoding size but remains restricted. Other methods, such as quantum abstract interpretation [22, 19], offer automated analysis but rely on over-approximation, potentially reducing precision in bug detection.

- No approaches currently support regular model checking or parameterized verification for quantum systems. While some recent work [10] proposes algorithms for specific classes of parameterized quantum circuits, a general solution is still lacking. Bridging this gap requires the design and integration of model checking algorithms with abstraction techniques tailored to the mathematical and operational characteristics of quantum computation.

4 Conclusions and Outlook

Quantum circuit verification stands at a unique intersection of disciplines, bridging automata theory, quantum physics, and software engineering. This confluence not only enriches the methodological foundations of the field but also opens promising avenues for future research. A natural question is whether automata-based techniques, which have proven powerful for modeling and verifying quantum circuits, can be extended to other facets of quantum software engineering. For instance, can similar methods be adapted to reason about the more abstract and expressive constructs found in quantum programming languages?

Beyond detecting errors, the overarching goal of this line of work is to build confidence in quantum computing systems as they mature towards solving problems beyond the reach of classical computation. The challenges are substantial, stemming from the inherent complexity of quantum mechanics and the intricacies of quantum software, but so are the potential benefits. As quantum technologies advance, the demand for rigorous and scalable verification techniques will only intensify.

By harnessing the expressive power of automata theory and the precision of symbolic reasoning, we can make significant strides toward ensuring the reliability and correctness of quantum software. Such advances will be crucial for enabling the widespread adoption of quantum computing and realizing its transformative potential across science and technology.

References

1 Parosh Aziz Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012. doi:10.1007/s10009-011-0216-8.

2 Parosh Aziz Abdulla. A symbolic approach to verifying quantum systems, 2025. To appear. doi:10.1145/3725725.

3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, and Tuan Phong Ngo. The benefits of duality in verifying concurrent programs under TSO. In *CONCUR*, volume 59 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl, 2016. doi:10.4230/LIPICS.CONCUR.2016.5.

4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson, and Ahmed Rezine. Counter-example guided fence insertion under tso. In *TACAS*, volume 7214 of *LNCS*, pages 204–219. Springer, 2012. doi:10.1007/978-3-642-28756-5_15.

5 Parosh Aziz Abdulla, Ahmed Bouajjani, Lukás Holík, Lisa Kaati, and Tomás Vojnar. Computing simulations over tree automata. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 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 4963 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2008. doi:10.1007/978-3-540-78800-3_8.

6 Parosh Aziz Abdulla, Ahmed Bouajjani, Lukás Holík, Lisa Kaati, and Tomás Vojnar. Composed bisimulation for tree automata. *Int. J. Found. Comput. Sci.*, 20(4):685–700, 2009. doi:10.1142/S0129054109006814.

7 Parosh Aziz Abdulla, Ahmed Bouajjani, Lukás Holík, Lisa Kaati, and Tomás Vojnar. Composed bisimulation for tree automata. In *CIAA08, 13th International Conference on the Implementation and Applications of Automata*, 2008.

8 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 313–321. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561359.

9 Parosh Aziz Abdulla, Yo-Ga Chen, Yu-Fang Chen, Kai-Min Chung, Lukás Holík, Ondrej Lengál, Jyun-Ao Lin, Fang-Yi Lo, Wei-Lun Tsai, and Di-De Yen. An automata-based framework for quantum circuit verification. 2025.

10 Parosh Aziz Abdulla, Yo-Ga Chen, Yu-Fang Chen, Lukás Holík, Ondrej Lengál, Jyun-Ao Lin, Fang-Yi Lo, and Wei-Lun Tsai. Verifying quantum circuits with level-synchronized tree automata. *Proc. ACM Program. Lang.*, 9(POPL):923–953, 2025. doi:10.1145/3704868.

11 Parosh Aziz Abdulla, Johanna Högberg, and Lisa Kaati. Bisimulation minimization of tree automata. *Int. J. Found. Comput. Sci.*, 18(4):699–713, 2007. doi:10.1142/S0129054107004929.

12 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking.*, pages 685–725. Springer, 2018. doi:10.1007/978-3-319-10575-8_21.

13 Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. symqv: Automated symbolic verification of quantum programs. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pages 181–198. Springer, 2023. doi:10.1007/978-3-031-27481-7_12.

14 Yu-Fang Chen, Kai-Min Chung, Ondrej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. An automata-based framework for verification and bug hunting in quantum circuits. *Proc. ACM Program. Lang.*, 7(PLDI):1218–1243, 2023. doi:10.1145/3591270.

15 Yu-Fang Chen, Philipp Rümmer, and Wei-Lun Tsai. A theory of cartesian arrays (with applications in quantum circuit verification). In Brigitte Pientka and Cesare Tinelli, editors, *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, July 1-4, 2023, Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2023. doi:10.1007/978-3-031-38499-8_10.

16 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. 2008. URL: https://inria.hal.science/hal-03367725.

17 Ellie D'Hondt and Prakash Panangaden. Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3):429–451, 2006. doi:10.1017/S0960129506005251.

18 James C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7), 1976. doi:10.1145/360248.360252.

19 Simon Perdrix. Quantum entanglement analysis based on abstract interpretation. In *International Static Analysis Symposium*, pages 270–282. Springer, 2008. doi:10.1007/978-3-540-69166-2_18.

20 Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, USA, 1 edition, 2008.

21 Mingsheng Ying. Floyd-Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6):1–49, 2012. doi:10.1145/2049706.2049708.

22 Nengkun Yu and Jens Palsberg. Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 542–558, 2021. doi:10.1145/3453483.3454061.

Unboundedness Problems for Formal Languages

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

Abstract

Informally, unboundedness problems are decision problems that ask about the existence of infinitely many words (satisfying certain properties) in a formal language. For example: Is a given language infinite? Or: Does a given language have super-polynomial growth? These came into focus in recent years because of their connections to downward closure computation and separability problems. Although unboundedness problems may seem difficult at first, it turns out that there are techniques that are at the same time conceptually very simple, but also apply to a surprisingly wide variety of language classes. The talk will survey recent results (and techniques) concerning unboundedness problems.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Decidability, formal languages, unifying frameworks, downward closure, separability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.2

Category Invited Talk

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

A classic theme of automata theory is to understand, given (i) a particular model of automata and (ii) a decision problem, whether this decision problem is decidable for this model. For example, the emptiness problem is decidable for the model of non-deterministic automata. Sometimes, it is even possible to obtain *general decidability results*: This means, they are of one of the following forms:

Type I: Class of models For a class of models, the result shows decidability of a particular problem.

Type II: Class of problems For a class of decision problems, the result shows decidability for a particular model of automata.

A well-known example of a result of type I is the fact that for well-structured transition systems (under mild assumptions), the coverability and termination problem are decidable [2, 34]. However, in this talk, we will focus on *unboundedness problems*: This is not a precise notion, but intuitively, these are problems concerning formal languages (represented by some automaton) that ask about the existence of *infinitely many words*.

The following are examples of unboundedness problems:

Infinity Given a language $L \subseteq \Sigma^*$ (represented by some automaton), does L contain infinitely many words?

Downward closure computation A classic result of Higman [38] and Haines [37] is that the set $L{\downarrow}$ of (scattered) subwords of any language $L \subseteq \Sigma^*$ is regular. The problem of downward closure computation is, given a language $L \subseteq \Sigma^*$, compute an NFA for $L{\downarrow}$.

© Georg Zetzsche;
licensed under Creative Commons License CC-BY 4.0

Boundedness A language $L \subseteq \Sigma^*$ is *bounded* if there are words $w_1, \ldots, w_n \in \Sigma^*$ with $L \subseteq w_1^* \cdots w_n^*$. The boundedness problem asks, given a language $L \subseteq \Sigma^*$, whether L is bounded.

Here, downward closure computation is an unboundedness problem, because e.g. determining that $L{\downarrow} = a^*$ means in particular concluding that L contains arbitrarily many occurrences of a. Moreover, concluding that a language L is not bounded means proving in particular that L is infinite.

We focus on unboundedness problems for two reasons. On the one hand, they have attracted significant attention in recent years: For example, computing downward closures is often useful for the verification for complex types of automata (see, e.g. [7,12,13,15,16,45,52]). Another application is separability problems, which have been studied intensively in recent years, either as separability by PTL [31,35] (see below for a definition), regular languages [8,14, 17,20,24,29,30,32,32,40–43,59], or other kinds of languages [1,18,21,23,54–56]. Separability problems are closely intertwined with unboundedness problems (see, e.g. [31]) and can be viewed as unboundedness problems themselves. The second reason is that unboundedness problems have turned out to be particularly well-suited for general results as above.

Here, we will briefly outline the results mentioned in the talk.

2 Downward closures, PTL separability, and simultaneous unboundedness

Our first result is about computing downward closures, and separability by piecewise testable languages (PTL). The former result is explained above. The PTL separability problem asks, given languages $K, L \subseteq \Sigma^*$ whether there exists a piecewise testable language S such that $K \subseteq S$ and $L \cap S = \emptyset$. Here, a language $S \subseteq \Sigma^*$ is *piecewise testable* if it is a Boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots a_n \Sigma^*$, where $n \geq 0$, $a_1, \ldots, a_n \in \Sigma$.

For many models of automata, their recognized class of languages is closed under *rational transductions*: Those are relations described by automata, called *transducers*, with one input tape and one output tape. A run of the transducer thus reads an input word and produces an output word. Applying the transducer to a language L then yields a new language consisting of all ouput words of transducer runs that read a word from L.

In this result, the model of automata is represented by its *language class*, i.e. the collection of languages recognized by automata of our model. Such a language class $\mathcal{C}$ is a *full trio* if for every language L from $\mathcal{C}$, applying a transducer to L will again yield a language from $\mathcal{C}$.

Finally, our result involves the *simultaneous unbounded problem* (SUP), which asks, given a language $L \subseteq a_1^* \cdots a_n^*$, whether for every $k \geq 0$, there is a word $a_1^{x_1} \cdots a_n^{x_n} \in L$ such that $x_1, \ldots, x_n \geq k$.

We are ready to state the first result, which characterizes computability of downward closures and decidability of PTL separability, by the much simpler SUP. It is combination of a characterization of computable downward closures [60, Theorem 1] and a characterization of decidable PTL-separability due to Czerwiński, Martens, Rooijen, Zeitoun, and the author [31, Theorem 2.6]:

▶ **Theorem 2.1.** *For every full trio $\mathcal{C}$, the following are equivalent:*

1. *Downward closures are computable for $\mathcal{C}$.*
2. *PTL separability is decidable for $\mathcal{C}$.*
3. *The SUP is decidable for $\mathcal{C}$.*

Motivated by this result, the SUP was then shown to be decidable for indexed languages [60], more generally for higher-order pushdown automata [36], and even for more generally for higher-order recursion schemes [22] (see also [10, 53]). A similar approach was then also developed for computing tree downward closures of higher-order schemes [9].

3 Unboundedness predicates for VASS

The second result is about a specific model, and a class of decision problems. The model is the prominent model of *vector addition systems with states* (VASS), which are automata equipped with multiple counters that (i) hold natural numbers and (ii) cannot be tested for zero during the run. Hence, the counters can only be incremented and decremented. In the end, a run is *accepting* if all counters are zero.

The class of decision problems is the class of "unboundedness predicates": An *unboundedness predicate* is a predicate $\mathfrak{p}$ on languages such that

1. If $K \subseteq L$ and $\mathfrak{p}(K)$ holds, then $\mathfrak{p}(L)$ holds as well.
2. If $\mathfrak{p}(F(KL))$ holds, then $\mathfrak{p}(F(K))$ or $\mathfrak{p}(F(L))$ holds.
3. If $\mathfrak{p}(K \cup L)$ holds, then $\mathfrak{p}(K)$ or $\mathfrak{p}(L)$ holds.

Here, $F(L) := \{v \in \Sigma^* \mid \exists u, w \in \Sigma^* \colon uvw \in L\}$ denotes the set of *factors* of a language $L \subseteq \Sigma^*$. Examples of such unboundedness predicates are:

Infinity the predicate $\mathfrak{p}_\infty$, where $\mathfrak{p}_\infty(L)$ if and only if L is infinite,

Not being bounded the predicate $\mathfrak{p}_{\neg \mathsf{b}}$, where $\mathfrak{p}_{\neg \mathsf{b}}(L)$ if and only if L is not bounded,

Factor universality the predicate $\mathfrak{p}_{\mathsf{funi}}$, where $\mathfrak{p}_{\mathsf{funi}}(L)$ if and only if every word in Σ^* appears as a factor in L

We say that an unboundedness predicate $\mathfrak{p}$ is *decidable* for a language class $\mathcal{C}$ if given a language L from $\mathcal{C}$, we can decide whether $\mathfrak{p}(F(L))$ holds. Hence, note that unboundedness predicates are always checked for the set of factors of the input language.

The following was shown by Czerwiński, Hofman, and the author [28, Theorem 3.1]:

▶ **Theorem 3.1.** *Let $\mathfrak{p}$ be an unboundedness predicate. Then $\mathfrak{p}$ is decidable for languages of VASS if and only if $\mathfrak{p}$ is decidable for regular languages.*

In fact, [28, Theorem 3.1] is stronger, since it is about a more general notion of unboundedness predicate, which also generalizes the SUP. In [11, Theorem 2], the authors show a version of Theorem 3.1 with complexity guarantees for automata with integer counters (i.e. which can attain negative values) and with/without a pushdown store.

4 Amalgamation

Our next result provides a sufficient condition for an automata model to have decidability of a particular set of decision problems. This sufficient condition is called *(concatenative) amalgamation*. The result is in the same spirit as the classical results about well-structured transition systems: In the latter, one assumes that the set of *configurations* in an automaton is equipped with a well-quasi ordering such that the reachability relation is (in some particular sense) compatible with the ordering. This leads to several decidability results [2, 30, 34].

In contrast, in *(concatenative) amalgamation systems*, one assumes a well-quasi ordering $\trianglelefteq$ on the set of *runs* of an automaton. Here if $\rho_0 \trianglelefteq \rho_1$, then the word read by ρ_0 must be a subword of the word read by ρ_1. Moreover, whenever one has runs ρ_0, ρ_1, ρ_2 such that $\rho_0 \trianglelefteq \rho_1$ and $\rho_0 \trianglelefteq \rho_2$, then there is a fourth run ρ_3 with $\rho_1 \trianglelefteq \rho_3$ and $\rho_2 \trianglelefteq \rho_3$ such that, roughly speaking, the word of ρ_3 is obtained by concatenating the gap words between ρ_0 and

ρ_1 and between ρ_0 and ρ_2. We say that a language class *has (concatenative) amalgamation* if each of its languages is recognized by a concatenative amalgamation system. For a detailed definition, see [5].

The idea for this notion comes from the theory of VASS: Jančar has introduced a well-quasi ordering on runs [39]. Based on ideas of Leroux [48], Leroux and Schmitz [50] observed that runs in VASS can be amalgamated as above. In [49], this was extended to pushdown VASS, which are VASS that also have access to a pushdown store. The amalgamation technique has then been used in structural [33, 49] and in algorithmic [21, 32] results on (pushdown) VASS.

Amalgamation as an abstract notion. The paper [5] proposes to view amalgamation as an abstract property that an automata model can satisfy. The paper shows that amalgamation is not only available in (pushdown) VASS, but in a hierarchy of infinite-state automata beyond pushdown VASS. In particular, all models in the unifying framework of valence automata [61] with decidable emptiness satisfy the amalgamation property.

Moreover, the paper [5] shows that many unboundedness problems can actually be shown decidable using only amalgamation, and with *surprisingly simple proofs*. The main algorithmic result of that paper [5, Main Theorem A] is the following:

▶ **Theorem 4.1.** *If $\mathcal{C}$ is a full trio with concatenative amalgamation, then the following are equivalent:*

1. *The SUP is decidable.*
2. *Downward closures are computable.*
3. *Separability by piecewise testable languages is decidable.*
4. *Boundedness is decidable.*
5. *Languages over one letter are effectively regular.*
6. *Priority downward closures are computable.*
7. *Emptiness is decidable.*

Here, priority downward closures are a regular overapproximation of languages that preserves more information than downward closures [6].

5 Bounded treewidth, multiple context-free languages, and downward closures

Finally, let us briefly mention some very recent work [3]. It concerns the notion of *bounded-treewidth systems*, which is a unifying framework for deciding *reachability* (and other properties) of infinite-state automata.

A seminal paper by Madhusudan and Parlato [51], together with a line of follow-up work [4, 26, 27, 46] revealed that many models of infinite-state automata (particularly those obtained as underapproximations of multi-pushdown automata) are *bounded-treewidth* [19, 44, 47, 57]. This means, the executions can be represented as graphs that have small tree decompositions. Therefore, decidability of these models can be uniformly explained by Courcelle's Theorem [25], which says that among structures of bounded treewidth, satisfiability of any property definable in monadic second-order logic (MSO) is decidable in polynomial time.

While this uniform framework is useful for yes-or-no queries like reachability, it yields little information about languages.

The results in [3] bring some light into this. It analyzes the languages in the slightly smaller class of *bounded-special-treewidth systems*: Bounded special treewidth is a somewhat stronger restriction that bounded treewidth, but coincides with bounded treewidth on all bounded-degree graphs. In fact, all bounded-treewidth systems studied in the literature have in fact bounded special treewidth.

It is shown in [3] that the languages of bounded-special-treewidth systems are exactly those generated by *multiple context-free grammars* (MCFG), a formalism originally developed in computational linguistics [58]. Intuitively, in an derivation of an MCFG, a non-terminal generates a tuple of words rather than a single word as in a context-free grammar.

In addition, it is shown in [3] that the downward closure of MCFG is computable in doubly exponential time (and that this is asymptotically optimal). Hence, this yields a downward closure computation algorithm for all bounded-special-treewidth systems.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the separability problem of string constraints. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 16:1–16:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CONCUR.2020.16.

2 Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000. doi:10.1006/inco.1999.2843.

3 C. Aiswarya, Pascal Baumann, Prakash Saivasan, Lia Schütze, and Georg Zetzsche. Bounded treewidth, multiple context-free grammars, and downward closures, 2026. To appear.

4 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014. doi:10.1007/978-3-319-11936-6_1.

5 Ashwani Anand, Sylvain Schmitz, Lia Schütze, and Georg Zetzsche. Verifying unboundedness via amalgamation. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 4:1–4:15. ACM, 2024. doi:10.1145/3661814.3662133.

6 Ashwani Anand and Georg Zetzsche. Priority downward closures. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 39:1–39:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.39.

7 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:4)2011.

8 Pablo Barceló, Diego Figueira, and Rémi Morvan. Separating automatic relations. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.17.

9 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Pawel Parys. Cost automata, safe schemes, and downward closures. *Fundam. Informaticae*, 188(3):127–178, 2022. doi:10.3233/FI-222145.

10 David Barozzini, Pawel Parys, and Jan Wroblewski. Unboundedness for recursion schemes: A simpler type system. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 112:1–112:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.112.

11 Pascal Baumann, Flavio D'Alessandro, Moses Ganardi, Oscar H. Ibarra, Ian McQuillan, Lia Schütze, and Georg Zetzsche. Unboundedness problems for machines with reversal-bounded

counters. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 240–264. Springer, 2023. doi:10.1007/978-3-031-30829-1_12.

12 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Checking refinement of asynchronous programs against context-free specifications. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 110:1–110:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.110.

13 Pascal Baumann, Moses Ganardi, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded analysis of concurrent programs (invited talk). In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.3.

14 Pascal Baumann, Eren Keskin, Roland Meyer, and Georg Zetzsche. Separability in Büchi VASS and singly non-linear systems of inequalities. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 126:1–126:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.126.

15 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. The complexity of bounded context switching with dynamic thread creation. 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 111:1–111:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.111.

16 Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. Context-bounded verification of thread pools. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022. doi:10.1145/3498678.

17 Pascal Baumann, Roland Meyer, and Georg Zetzsche. Regular separability in Büchi VASS. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.9.

18 Mikolaj Bojanczyk. It is undecidable if two regular tree languages can be separated by a deterministic tree-walking automaton. *Fundam. Informaticae*, 154(1-4):37–46, 2017. doi:10.3233/FI-2017-1551.

19 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

20 Lorenzo Clemente, Wojciech Czerwiński, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 117:1–117:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2017.117.

21 Lorenzo Clemente, Wojciech Czerwiński, Slawomir Lasota, and Charles Paperman. Separability of reachability sets of vector addition systems. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017,*

Hannover, Germany, volume 66 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.STACS.2017.24.

22 Lorenzo Clemente, Pawel Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

23 Lorenzo Clemente and Michal Skrzypczak. Deterministic and game separability for regular languages of infinite trees. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 126:1–126:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.126.

24 Elias Rojas Collins, Chris Köcher, and Georg Zetzsche. The complexity of separability for semilinear sets and parikh automata. In Pawel Gawrychowski, Filip Mazowiecki, and Michal Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, August 25-29, 2025, Warsaw, Poland*, volume 345 of *LIPIcs*, pages 38:1–38:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.MFCS.2025.38.

25 Bruno Courcelle. The monadic second-order logic of graphs, II: infinite graphs of bounded width. *Math. Syst. Theory*, 21(4):187–221, 1989. doi:10.1007/BF02088013.

26 Aiswarya Cyriac. *Verification of communicating recursive programs via split-width. (Vérification de programmes récursifs et communicants via split-width)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: https://tel.archives-ouvertes.fr/tel-01015561.

27 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012. doi:10.1007/978-3-642-32940-1_38.

28 Wojciech Czerwinski, Piotr Hofman, and Georg Zetzsche. Unboundedness problems for languages of vector addition systems. 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 119:1–119:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.119.

29 Wojciech Czerwiński and Slawomir Lasota. Regular separability of one counter automata. 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.2017.8005079.

30 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CONCUR.2018.35.

31 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discret. Math. Theor. Comput. Sci.*, 19(4), 2017. doi:10.23638/DMTCS-19-4-1.

32 Wojciech Czerwinski and Georg Zetzsche. An approach to regular separability in vector addition systems. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 341–354. ACM, 2020. doi:10.1145/3373718.3394776.

33 Alain Finkel, Shankara Narayanan Krishna, Khushraj Madnani, Rupak Majumdar, and Georg Zetzsche. Counter machines with infrequent reversals. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSTTCS.2023.42.

34 Alain Finkel and Ph Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

35 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. 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 97:1–97:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICALP.2016.97.

36 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL 2016: Principles of Programming Languages*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.

37 Leonard H Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi:10.1016/S0021-9800(69)80111-0.

38 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of The London Mathematical Society*, pages 326–336, 1952. doi:10.1112/PLMS/S3-2.1.326.

39 Petr Jancar. Decidability of a temporal logic problem for Petri nets. *Theor. Comput. Sci.*, 74(1):71–93, 1990. doi:10.1016/0304-3975(90)90006-4.

40 Eren Keskin and Roland Meyer. Separability and non-determinizability of WSTS. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.8.

41 Eren Keskin and Roland Meyer. On the separability problem of VASS reachability languages. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 49:1–49:14. ACM, 2024. doi:10.1145/3661814.3662116.

42 Chris Köcher and Georg Zetzsche. Regular separators for VASS coverability languages. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.FSTTCS.2023.15.

43 Eryk Kopczynski. Invisible pushdown languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 867–872. ACM, 2016. doi:10.1145/2933575.2933579.

44 Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 161–170. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.9.

45 Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 72–84. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.CONCUR.2015.72.

46 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. A unifying approach for multistack pushdown automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International*

Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I, volume 8634 of *Lecture Notes in Computer Science*, pages 377–389. Springer, 2014. doi:10.1007/978-3-662-44522-8_32.

47 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Reachability of scope-bounded multistack pushdown systems. *Inf. Comput.*, 275:104588, 2020. doi:10.1016/J.IC.2020.104588.

48 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. 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 307–316. ACM, 2011. doi:10.1145/1926385.1926421.

49 Jérôme Leroux, M. Praveen, Philippe Schnoebelen, and Grégoire Sutre. On functions weakly computable by pushdown Petri nets and related systems. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:15)2019.

50 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

51 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. *SIGPLAN Not.*, 46(1):283–294, 2011. doi:10.1145/1925844.1926419.

52 Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetzsche. General decidability results for asynchronous shared-memory programs: Higher-order and beyond. *Log. Methods Comput. Sci.*, 18(4), 2022. doi:10.46298/LMCS-18(4:2)2022.

53 Pawel Parys. The complexity of the diagonal problem for recursion schemes. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.45.

54 Thomas Place and Marc Zeitoun. Separation for dot-depth two. *Log. Methods Comput. Sci.*, 17(3), 2021. doi:10.46298/LMCS-17(3:24)2021.

55 Thomas Place and Marc Zeitoun. A generic polynomial time approach to separation by first-order logic without quantifier alternation. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 43:1–43:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.FSTTCS.2022.43.

56 Thomas Place and Marc Zeitoun. Group separation strikes back. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–13. IEEE, 2023. doi:10.1109/LICS56636.2023.10175683.

57 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005. doi:10.1007/978-3-540-31980-1_7.

58 Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theor. Comput. Sci.*, 88(2):191–229, 1991. doi:10.1016/0304-3975(91)90374-B.

59 Ramanathan S. Thinniyam and Georg Zetzsche. Regular separability and intersection emptiness are independent problems. 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 51:1–51:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSTTCS.2019.51.

60 Georg Zetzsche. An approach to computing downward closures. 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 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

61 Georg Zetzsche. *Monoids as Storage Mechanisms*. PhD thesis, Kaiserslautern University of Technology, Germany, 2016. URL: https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-44003.

PDQMA = DQMA = NEXP: QMA with Hidden Variables and Non-Collapsing Measurements

Scott Aaronson
University of Texas at Austin, TX, USA

Sabee Grewal
University of Texas at Austin, TX, USA

Vishnu Iyer
University of Texas at Austin, TX, USA

Simon C. Marshall
Leiden University, The Netherlands

Ronak Ramachandran
University of Texas at Austin, TX, USA

Abstract

We define and study a variant of QMA (Quantum Merlin Arthur) in which Arthur can make multiple non-collapsing measurements to Merlin's witness state, in addition to ordinary collapsing measurements. By analogy to the class PDQP defined by Aaronson, Bouland, Fitzsimons, and Lee (2014), we call this class PDQMA. Our main result is that PDQMA = NEXP; this result builds on the PCP theorem and complements the result of Aaronson (2018) that PDQP/qpoly = ALL. While the result has little to do with quantum mechanics, we also show a more "quantum" result: namely, that QMA with the ability to inspect the entire history of a hidden variable is equal to NEXP, under mild assumptions on the hidden-variable theory. We also observe that a quantum computer, augmented with quantum advice and the ability to inspect the history of a hidden variable, can solve any decision problem in polynomial time.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Quantum computation theory; Theory of computation → Quantum complexity theory; Theory of computation → Quantum complexity theory

Keywords and phrases quantum Merlin-Arthur, non-collapsing measurements, hidden-variable theories

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.3

Related Version *Full Version*: https://arxiv.org/abs/2403.02543

Funding SG and RR are supported via SA from the same funding sources.
Scott Aaronson: SA is supported by a Vannevar Bush Fellowship by the US Department of Defense, the Berkeley NSF-QLCI CIQC Center, a Simons Investigator Award, and the Simons "It from Qubit" collaboration.
Vishnu Iyer: VI is supported by an NSF Graduate Research Fellowship.
Simon C. Marshall: SCM is supported by the European Union's Horizon 2020 program (NEQASQC) and a gift from Google Quantum AI.

Acknowledgements SCM thanks David Dechant for useful discussions. SG, VI, and RR thank Joshua Cook, Siddhartha Jain, and Kunal Marwaha for helpful conversations.

1 Introduction

To understand the power of quantum computation is, in large part, to understand how that power depends on the central features of quantum mechanics itself, such as linearity, unitarity, tensor products, complex numbers, the Born Rule, or the destructive nature of measurement.


© Scott Aaronson, Sabee Grewal, Vishnu Iyer, Simon C. Marshall, and Ronak Ramachandran;
licensed under Creative Commons License CC-BY 4.0

But since quantum mechanics is usually presented as a "package deal," how can we pick apart these dependencies? One natural approach has been to define complexity classes based on "fantasy" versions of quantum mechanics, which change one or more of its features, and see how they relate to the standard BQP (Bounded-Error Quantum Polynomial-Time). Some representative findings of that research program over the past few decades include:

(1) Quantum mechanics over the reals or quaternions leads to the same computational power as quantum mechanics over the complex numbers, despite the theories differing in other respects [25].

(2) Quantum mechanics with a nonlinear Schrödinger equation would generically allow NP- and even #P-complete problems to be solved in polynomial time, in contrast to what is conjectured for standard (linear) quantum mechanics [11].

(3) A quantum computer with closed timelike curves could solve exactly the problems in PSPACE, same as a classical computer with closed timelike curves [10].

(4) Quantum computers with nonunitary linear evolution, or modifications of the Born rule (say, $|\psi|^3$), with normalization of probabilities imposed, yield at least the power of quantum computers with *postselected measurement outcomes* – a model that Aaronson called PostBQP and proved to coincide with the classical complexity class PP [2, 5].

(5) Generalized probabilistic theories (GPTs), a class of theories that includes quantum mechanics and classical probability theory as special cases, are as a whole characterized by the complexity class AWPP [23, 15].

(6) If we allow multiple, non-collapsing measurements of the same state – or, closely related, to see the entire history of a hidden variable as in Bohmian mechanics – we get a model of computation that seems more powerful than standard quantum computation, but only "slightly" so [4]. As examples, we can quickly find collisions in many-to-one functions (and thus, for example, solve Graph Isomorphism), and we can solve the Grover search problem in $N^{1/3}$ steps rather than $\sqrt{N}$. But we still seem unable to solve NP-complete problems in polynomial time.

Example (6) is the one of most interest to us here. To our knowledge, it is the only natural example known where changing the rules of quantum mechanics leads to complexity classes that appear only modestly larger than BQP. Much of the power comes from the combination of non-collapsing measurements with ordinary collapsing ones. As an example, given a two-to-one function $f : [N] \rightarrow [M]$, the way to find collisions is simply to prepare

$$\frac{1}{\sqrt{N}} \sum_{x \in [N]} |x\rangle |f(x)\rangle,$$

then measure the $|f(x)\rangle$ register in the ordinary way to get

$$\frac{|x\rangle + |y\rangle}{\sqrt{2}}$$

where $f(x) = f(y)$ in the first register, and finally perform multiple non-collapsing measurements on the first register in the standard basis, until both x and y are observed with high probability.[1]

[1] The reason why non-collapsing measurement allows Grover search in $N^{1/3}$ steps is simpler and does not require us to combine collapsing with non-collapsing measurements. Instead, given a unique marked item out of N, one simply runs Grover's algorithm for $T = N^{1/3}$ iterations, thereby boosting the probability of the marked item to $\sim \frac{T^2}{N} = N^{-1/3}$, and then performs $N^{1/3}$ non-collapsing measurements so that the marked item is found with constant probability.

In a hidden-variable theory, where the hidden variable is either at $|x\rangle$ or $|y\rangle$ with equal probabilities, the solution at this point is to "juggle" – for example, by repeatedly applying Fourier transforms followed by their inverses. The goal here is to cause the hidden variable to "forget" whether it was at $|x\rangle$ or $|y\rangle$, so that it must eventually visit both of them with high probability. In such a case, if (as we're imagining) we could see the whole history of the hidden variable at once, from some godlike vantage point, we would learn both $|x\rangle$ and $|y\rangle$ and thereby solve our computational problem.

The history of these ideas is a bit tangled. In 2005, Aaronson defined the class DQP (Dynamical Quantum Polynomial-Time), to capture the problems that quantum computers could efficiently solve, if only one could examine the entire history of a hidden variable (in any hidden-variable theory that satisfies reasonable axioms, called robustness and indifference). He showed that SZK $\subseteq$ DQP, where SZK is Statistical Zero Knowledge, basically because DQP could simulate non-collapsing measurements (although he didn't formalize this). Combined with Aaronson's quantum lower bound for finding collisions [1], which implies the existence of an oracle relative to which SZK $\not\subset$ BQP, this gives us an oracle separation between BQP and DQP. Aaronson also showed that DQP $\subseteq$ EXP, which has not been improved since then. He claimed to give an oracle relative to which NP $\not\subset$ DQP, although his proof had a bug [8] and the existence of such an oracle remains open. Then, in 2014, Aaronson, Bouland, Fitzsimons, and Lee [8] defined the class PDQP (Product Dynamical Quantum Polynomial-Time), to capture non-collapsing measurements specifically. They showed that PDQP also contains SZK, showed the upper bound PDQP $\subseteq$ $\mathsf{BPP}^{\mathsf{PP}}$, and gave a correct proof that there exists an oracle relative to which NP $\not\subset$ PDQP.

Overall, as we said, these results painted a picture of DQP and PDQP as only modestly more powerful than BQP. However, a surprising new wrinkle came in 2018, when Aaronson [6] observed that PDQP/qpoly = ALL, where /qpoly means "with polynomial-sized quantum advice," and ALL is the class of all languages. This stands in contrast to 2004 results of Aaronson [3] that limit the power of BQP/qpoly: namely, that BQP/qpoly $\subseteq$ PP/poly (later improved by Aaronson and Drucker [9] to BQP/qpoly $\subseteq$ QMA/poly), and that there exists an oracle relative to which NP $\not\subset$ BQP/qpoly. In other words, quantum advice and non-collapsing measurements have a "Mentos and Coke" character, where each one is only modestly powerful in isolation, but together they trigger a complexity-theoretic explosion.

To prove the PDQP/qpoly = ALL result, Aaronson adapted a 2005 theorem of Raz [24] that QIP/qpoly = ALL, where QIP is the class of languages that admit quantum interactive proofs. In both Raz's protocol and Aaronson's, given an arbitrary Boolean function $f : \{0,1\}^n \to \{0,1\}$ that one wants to compute, one first chooses a prime $q \gg n$. The whole truth table of f is then encoded by the quantum advice state

$$|\psi\rangle = \frac{1}{\sqrt{q^n}} \sum_{z \in \mathbb{F}_q^n} |z\rangle|p(z)\rangle,$$

where $p : \mathbb{F}_q^n \to \mathbb{F}_q$ is the unique multilinear polynomial over $\mathbb{F}_q$ such that $p(x) = f(x)$ for all $x \in \{0,1\}^n$. Next, given a point of interest $x \in \{0,1\}^n$, on which one wants to evaluate $f(x)$, one measures $|\psi\rangle$ so as to collapse it to an equal superposition

$$|\psi_\ell\rangle = \frac{1}{\sqrt{q-1}} \sum_{z \in \ell \setminus \{x\}} |z\rangle|p(z)\rangle$$

over a random line $\ell \subset \mathbb{F}_q^n$ that passes through x, minus the point x itself. Finally, one uses polynomial interpolation on this line to recover $p(x) = f(x)$. In the non-collapsing measurements model, this is done by simply measuring the state $|\psi_\ell\rangle$ over and over in the standard basis, until enough $(z, p(z))$ pairs have been observed for the interpolation to work.[2]

1.1 This Paper

Here we show a new example where non-collapsing measurements combined with one other resource yield extraordinary computational power – vastly more power than either resource in isolation.

The class QMA (Quantum Merlin Arthur) is a well-known quantum generalization of NP; it consists of all languages for which a "yes" answer can be verified in quantum polynomial time with the help of a polynomial-size quantum witness state. We define and study PDQMA (Product Dynamical QMA), or QMA augmented with non-collapsing measurements. Our main result is that PDQMA = NEXP, even if PDQMA is defined with a completeness/soundness gap of (say) $1 - 2^{-n}$ vs. 2^{-n} (Theorem 5).

Since the inclusion PDQMA $\subseteq$ NEXP is straightforward, the interesting part is NEXP $\subseteq$ PDQMA. By the celebrated PCP theorem [14, 13], it suffices to show that a two-query PCP system (Definition 3) can be simulated by PDQMA. Our proof builds on Aaronson's proof [6] that PDQP/qpoly = ALL, but with two key differences. First, to simulate a two-query PCP system, we need a witness state that can support at least *two* queries to an exponentially long truth table, rather than just one query. Second, we now need an analysis of *soundness*: why, for example, can the prover not cheat by sending a witness that causes the response to each query to depend on the other query? This problem seems particularly acute once we realize that we can no longer rely on the no-signaling principle of quantum mechanics when non-collapsing measurements are allowed. To address the first challenge, we extend Aaronson's algorithm to support multiple queries to an exponentially long truth table. To address the second challenge, we show that one can use non-collapsing measurements to perform a low-degree test, ensuring that Merlin's witness has the "error-correcting code" structure of an honest witness state.

Let us highlight our simulation of a classical PCP as additional motivation for our work. Specifically, simulating PCP systems is one of the few approaches to show that quantum Merlin-Arthur systems can solve NEXP (see e.g. [7, 22]). An important direction for future work is to adapt our simulation to quantum Merlin-Arthur systems that adhere to the laws of quantum mechanics (even if the verifier is allowed exponential time or to interact with the prover). We view any progress of this form as progress toward understanding more "reasonable" quantum Merlin-Arthur systems, such as QMA(2).

We point out some implications of our result. First, combining with the Nondeterministic Time Hierarchy Theorem (NP $\neq$ NEXP), we find that PDQMA is *unconditionally* more powerful than NP. Second, "scaling down by an exponential," we find that when non-collapsing measurements are allowed, the problem of optimizing an acceptance probability over all N-dimensional quantum states jumps from easy (a principal eigenvector problem) to NP-hard even to approximate.

Let us place our result in the context of previous work on generalizations of QMA. Aharonov and Regev [12] defined QMA+, a variant of QMA where the verifier can directly obtain the probability a given two-outcome measurement will accept, and showed QMA+ =

[2] Since p is a multilinear extension of a Boolean function on n variables, its degree is at most n. Hence, $n + 1$ pairs are needed to do polynomial interpolation, so q must be chosen to be at least $n + 2$.

QMA. More recently, Jeronimo and Wu [22] showed that $\mathsf{QMA}^+(2) = \mathsf{NEXP}$, where $\mathsf{QMA}^+(2)$ is QMA with two unentangled proofs that have nonnegative real amplitudes. Bassirian, Fefferman, and Marwaha [16] improved this to show that $\mathsf{QMA}^+ = \mathsf{NEXP}$ – i.e., nonnegative amplitudes alone suffice for the jump to NEXP. As of this writing, it remains a matter of avid speculation whether unentangled witnesses *also* suffice for the jump to NEXP: that is, whether $\mathsf{QMA}(2) = \mathsf{NEXP}$. Of course, our result implies that it would suffice to simulate non-collapsing measurements using unentangled proofs – i.e., to show $\mathsf{PDQMA} \subseteq \mathsf{QMA}(2)$.

One important observation about our PDQMA protocol is that it only ever measures the witness state in the computational basis – and hence, one could say, never exploits quantum interference. So in particular, if we defined a complexity class PDMA (Product Dynamical Merlin-Arthur) in mathematical parallel to PDQMA, where the witness was a classical probability distribution $\mathcal{D}$, and one was allowed to sample from $\mathcal{D}$ with or without doing Bayesian updating to it, we would equally have $\mathsf{PDMA} = \mathsf{NEXP}$. The main difference here is simply that it seems hard to invent a story that motivates PDMA.

In Section 4, we sharpen this point by defining and studying a class that we call DQMA (Dynamical QMA), or QMA augmented by the ability to see the entire history of a hidden variable. In particular, the verifier can perform a DQP (Dynamical Quantum Polynomial-Time) computation, a complexity class introduced by Aaronson to study hidden variable theories from a quantum computing perspective [4]. In short, DQP captures computations where one evolves a state vector unitarily, but there is a deeper "hidden variable" in a definite state that evolves stochastically. At the end of the computation, the history of the hidden variable (i.e., the sequence of definite states it occupied at each step) can be viewed when making the final accept/reject decision. DQP captures any hidden variable theory that satisfies a set of axioms (given in Section 4), including theories due to Dieks [19] and Schrödinger [26].

We show that $\mathsf{DQMA} = \mathsf{NEXP}$ (Theorem 16). Here the reasons really do depend on quantum mechanics. Specifically, they depend on the ability to "hide" crucial information in the phases of amplitudes, to prevent a hidden variable trajectory from remembering that information. If we tried to define the analogous class DMA (Dynamical MA), it would trivially coincide with MA. Assuming $\mathsf{MA} = \mathsf{NP}$, as follows from a standard derandomization assumption, this has the amusing consequence that $\mathsf{DMA} \neq \mathsf{DQMA}$ – that is, in the presence of both witness states and trajectory sampling, quantum can already be known to be stronger than classical. In the same section, we also observe that our techniques imply $\mathsf{DQP/qpoly} = \mathsf{ALL}$ (Theorem 17), complementing Aaronson's result that $\mathsf{PDQP/qpoly} = \mathsf{ALL}$ [6]. This result relies on quantum mechanics in the same way that our DQMA result does.

One might also wonder about other "fantasy" variants of QMA. In principle, any variant of BQP can be combined with QMA to yield a new complexity class. For example, consider a variant of BQP that can clone states (i.e., perform the transformation $|\psi\rangle|0^n\rangle \mapsto |\psi\rangle^{\otimes 2}$). It is easy to see that one can simulate k non-collapsing measurements of a state by cloning the state k times and then measuring each copy in the usual collapsing way. Hence, combining this BQP variant with QMA yields a complexity class equal to NEXP by our Theorem 5. Indeed, if one wants to alter Arthur's powers *without* triggering a "Mentos and Coke" effect, they would need to find a BQP variant that is only modestly more powerful, the way PDQP is.

1.2 Main Ideas

We give a high-level overview of our proof that $\mathsf{PDQMA} = \mathsf{NEXP}$ (Theorem 5). Informally, PDQMA is like QMA except the verifier can perform non-collapsing measurements in addition to the normal collapsing ones (see Definition 2 for a formal definition). The containment $\mathsf{PDQMA} \subseteq \mathsf{NEXP}$ is straightforward because NEXP can guess the exponentially-long classical description of the PDQMA witness and verify it in exponential time.

Thus, the challenge is proving $\mathsf{NEXP} \subseteq \mathsf{PDQMA}$. We show this by simulating a two-query probabilistically checkable proof (PCP) system for NEXP (see Definition 3 and Theorem 4). In short, the celebrated PCP theorem [14, 13] tells us that languages in NEXP can be decided by a verifier that queries a PCP $\pi : \{0,1\}^n \to \Sigma$ at two points of the verifier's choosing, where Σ is a constant-sized alphabet. Hence, our result follows from simulating two queries to a π.

The honest witness is the same as in [6]:

$$|\psi\rangle = \frac{1}{\sqrt{q^n}} \sum_{z \in \mathbb{F}_q^n} |z\rangle |p(z)\rangle,$$

where $p : \mathbb{F}_q^n \to \mathbb{F}_q$ is the unique degree-n multilinear extension of the PCP $\pi : \{0,1\}^n \to \Sigma$ and $q = O(n)$ is chosen to be sufficiently large. However, our setting differs from [6] in two key ways: (i) we must retrieve two values $\pi(w)$ and $\pi(w')$ for our choice of w, w', rather than one, and (ii) the quantum witness can no longer be trusted (as it can be in the advice setting).

To explain how to handle the first difference, let us briefly recall how to simulate a single query. As discussed, given a point of interest $w \in \{0,1\}^n$, on which one wants to evaluate $\pi(w)$, Aaronson [6] measures $|\psi\rangle$ so as to collapse it to an equal superposition

$$|\psi_\ell\rangle = \frac{1}{\sqrt{q-1}} \sum_{z \in \ell \setminus \{w\}} |z\rangle |p(z)\rangle$$

over a random line $\ell \subset \mathbb{F}_q^n$ that passes through w, minus the point w itself. To recover two values $w, w' \in \{0,1\}^n$, one can generalize Aaronson's procedure so that the measurement on $|\psi\rangle$ collapses it to an equal superposition over an *affine plane* that contains the points w and w', minus the unique affine line containing w and w'. (Indeed, this can be generalized to any k-dimensional affine subspace, where the measurement collapses $|\psi\rangle$ to a superposition over a random k-dimensional affine subspace containing points $w_1, \ldots, w_k$ minus the unique $(k-1)$-dimensional affine subspace containing $w_1, \ldots, w_k$.) Then, just as in [6], one performs enough non-collapsing measurements to recover $\pi(w)$ and $\pi(w')$ via polynomial interpolation.

The only remaining issue is that Merlin can potentially cheat and send a quantum state that is far from the honest witness. To address this, we use the lines-point low-degree test (Lemma 1) given by Friedl and Sudan [20]. Recall that an affine line $\ell \subset \mathbb{F}_q^n$ passing through points $x, y \in \mathbb{F}_q^n$ is the set $\{x + (y-x)t\}_{t \in \mathbb{F}_q^n}$, so a function $g : \ell \to \mathbb{F}_q$ can be viewed as a univariate polynomial in t. The lines-point low-degree test says that if a function $f : \mathbb{F}_q^n \to \mathbb{F}_q$ restricted to a randomly chosen line agrees with a degree-n univariate polynomial, then f agrees with some degree-n polynomial $h : \mathbb{F}_q^n \to \mathbb{F}_q$ on all of $\mathbb{F}_q^n$.

Observe that to perform this low-degree test, we need to make sure the function that Merlin encodes into his witness agrees with a low-degree polynomial on a randomly chosen affine line. Therefore, rather than interpolating a polynomial on a random affine plane containing the points w and w' of interest, our verification procedure interpolates a polynomial on an *affine cube*. This affine cube contains the points $w, w' \in \{0,1\}^n$ but we also ensure that it contains a point $w'' \in \mathbb{F}_q^n$ that the verifier chooses uniformly at random. This ensures that the affine cube contains a uniformly random line independent of the points w and w'. Then, if the polynomial interpolation succeeds, we can (i) recover $\pi(w)$ and $\pi(w')$ as desired and (ii) conclude that Merlin's witness encoded a low-degree polynomial because it passed the lines-point low degree test (i.e., it agreed with a low-degree polynomial on a randomly chosen line).

To summarize, the procedure works as follows. First, Merlin commits to some witness state $|\psi\rangle$. Then the PDQMA verifier simulates the PCP verifier to obtain queries $w, w' \in \{0,1\}^n$ and picks a uniformly random $w'' \in \mathbb{F}_q^n$. The verifier measures Merlin's witness state $|\psi\rangle$ to collapse it to a superposition over points in a random affine cube containing w, w', and w'' minus the affine plane containing w, w', and w''. Then the verifier uses non-collapsing measurements to collect all $(z, p(z))$ pairs in the affine cube (minus the affine plane) and interpolates a polynomial $u : \mathbb{F}_q^3 \to \mathbb{F}_q$ that fits these pairs. If the polynomial interpolation succeeds, Merlin passes the lines-point low-degree test and the verifier can learn $\pi(w)$ and $\pi(w')$ by evaluating the polynomial u.

1.3 Concurrent Work

In concurrent and independent work, Bassirian and Marwaha [17] show that QMA with non-collapsing measurements equals NEXP. They observe that the proof of $\mathsf{QMA}^+ = \mathsf{NEXP}$ [16] goes through even if one replaces the promise of a non-negative witness state with the ability for the verifier to perform non-collapsing measurements. [16] can prove a containment (for a constant completeness/soundness gap) using a constant number of non-collapsing measurements, whereas our verification procedure always uses $O(n \log n)$ non-collapsing measurements. Our approach readily extends to prove $\mathsf{DQMA} = \mathsf{NEXP}$ and $\mathsf{DQP/qpoly} = \mathsf{ALL}$.

2 Preliminaries

Throughout this work, we use the following notation. $\mathbb{N} := \{1, 2, 3, \ldots\}$ denotes the natural numbers. For a finite field $\mathbb{F}_q$, $\mathbb{F}_q^* := \mathbb{F}_q \setminus \{0\}$.

Let $\mathcal{L}$ denote the set of all affine lines (i.e., 1-dimensional affine subspaces) of $\mathbb{F}_q^n$. Recall that the line passing through points $a, b \in \mathbb{F}_q^n$ is the set $\{a + (b-a)t\}_{t \in \mathbb{F}_q}$. Therefore, a polynomial g that maps a line $\ell \in \mathcal{L}$ to $\mathbb{F}_q$ can be thought of as a univariate polynomial. We will use the following lines-point low-degree test of Friedl and Sudan [20] (see also the restatement in [21, Theorem 1.2]).

▶ **Lemma 1** (Multivariate low-degree test [20]). *Let $f : \mathbb{F}_q^n \to \mathbb{F}_q$ be any n-variate function, and let $G = \{g_\ell\}_{\ell \in \mathcal{L}}$ be a collection of degree-d polynomials $g_\ell : \ell \to \mathbb{F}_q$. There is a constant C large enough such that for any d satisfying $q > Cd$, if*

$$\Pr_{\substack{\ell \sim \mathcal{L} \\ z \sim \ell}} [f(z) \neq g_\ell(z)] \leq \delta,$$

for some $0 < \delta < 0.01$, then there exists an n-variate degree-d polynomial h such that

$$\Pr_{z \sim \mathbb{F}_q^n} [f(z) \neq h(z)] \leq 4\delta.$$

In words, Lemma 1 is saying that, if a function f restricted to a randomly chosen line ℓ disagrees with a low-degree polynomial on at most a δ fraction of points, then there must exist a globally low-degree polynomial that f disagrees with on at most a 4δ fraction of points. In short, Lemma 1 gives a local way to test that the entire function is low degree.

This work also involves affine planes and cubes (i.e., 2- and 3-dimensional affine subspaces). Recall that an affine plane is uniquely defined by three independent points $a, b, c \in \mathbb{F}_q^n$. The plane containing these points is the set

$$\{a + (b-a)t_1 + (c-a)t_2\}_{t_1, t_2 \in \mathbb{F}_q}.$$

Similarly, the unique affine cube containing the independent points $a, b, c, d \in \mathbb{F}_q^n$ is the set

$$\{a + (b-a)t_1 + (c-a)t_2 + (d-a)t_3\}_{t_1,t_2,t_3 \in \mathbb{F}_q}.$$

We now give a formal definition of PDQMA.

▶ **Definition 2.** $\mathsf{PDQMA}_{c,s}$ *is the class of languages* $L \subseteq \{0,1\}^*$ *for which there exists a* PDQP *verifier* V *(to be defined shortly) such that, for all* $x \in \{0,1\}^*$*:*

- ***Completeness:*** *If* $x \in L$ *then there exists a witness state* $|\phi\rangle$*, on* $\mathrm{poly}(n)$ *qubits, such that* $V(x, |\phi\rangle)$ *accepts with probability at least* c*.*
- ***Soundness:*** *If* $x \notin L$ *then* $V(x, |\phi\rangle)$ *accepts with probability at most* s *for all witness states* $|\phi\rangle$*.*

A PDQP *verifier consists of two phases. In the first phase, a* P*-uniform quantum circuit* C_x*, depending on the input* x*, is applied to the initial state* $|\phi\rangle|0^{p(n)}\rangle$*, where* $|\phi\rangle$ *is the witness and* p *is a polynomial. This* C_x *can consist of three types of gates:* CNOT *gates, 1-qubit* $\pi/8$ *rotations, and measurements in the* $\{|0\rangle, |1\rangle\}$ *basis. The* CNOT *and* $\pi/8$ *gates provide universality for* BQP*, while the measurement gates introduce a probabilistic component. In the second phase, we consider the final state* $|\psi\rangle$ *of* C_x*, which depends in part on the probabilistic results of the measurement gates. Let* $\mathcal{D}$ *be the probability distribution induced by measuring all qubits of* $|\psi\rangle$ *in the computational basis and let* q *be a polynomial. Then a classical polynomial-time algorithm,* A*, receives as input* x *as well as* $q(n)$ *independent samples from* $\mathcal{D}$*, and then either accepts or rejects.*

Our proof that PDQMA = NEXP will rely on the characterization NEXP by probabilistically checkable proof (PCP) systems where the verifier only makes 2 queries to the PCP. Although this is well-known to classical complexity theorists, we explain this formally below for completeness. We begin by defining the complexity classes PCP.

▶ **Definition 3.** $\mathsf{PCP}_{c,s}[r,q]_\Sigma$ *is the class of languages* $L \subseteq \{0,1\}^*$ *for which there exists a probabilistic polynomial-time verifier* V_{PCP} *which uses* r *random bits and makes* q *queries to an oracle* π*, each time receiving a response in some alphabet* Σ *such that*

1. ***Completeness:*** *If* $x \in L$*, then* $\exists \pi$ *such that* $\Pr[V_{\mathsf{PCP}}^{\pi}(x) = 1] \geq c$*.*
2. ***Soundness:*** *If* $x \notin L$*, then* $\forall \pi$*,* $\Pr[V_{\mathsf{PCP}}^{\pi}(x) = 1] \leq s$*.*

The celebrated PCP theorem [14, 13] tells us that $\mathsf{NP} = \mathsf{PCP}_{1,s}[O(\log(n)), 3]_{\{0,1\}}$ for any constant s. A simple consequence is that $\mathsf{NEXP} = \mathsf{PCP}_{1,s}[\mathrm{poly}(n), 3]_{\{0,1\}}$. It is folklore that $\mathsf{PCP}_{c,s}[r,q]_\Sigma \subseteq \mathsf{PCP}_{c,s/q}[r + \log(q), 2]_{\Sigma^q}$, i.e., that the number of queries can be reduced to two at the cost of a larger alphabet and worse soundness error.

▶ **Theorem 4.** *For a size-*8 *alphabet* Σ *and any constant* $s \in (0,1)$*,* $\mathsf{PCP}_{1,s}[\mathrm{poly}(n), 2]_\Sigma = \mathsf{NEXP}$*.*

One can improve the soundness error to sub-constant by further increasing the alphabet size and our proofs that PDQMA = NEXP (Theorem 5) and DQMA = NEXP (Theorem 16) still go through with only slight (if any) modification.

3 QMA and Non-collapsing Measurements

We prove our main result: if QMA is modified so that Arthur can perform non-collapsing measurements in addition to standard quantum computation, then the resulting class equals NEXP.

The hard part is to show that NEXP $\subseteq$ PDQMA, which we prove by simulating a two-query PCP system for NEXP. At a high level, the PDQMA verifier is given a PCP $\pi : \{0,1\}^n \to \Sigma$ encoded in a quantum proof, and it suffices to learn π at two points of the verifier's choosing.

Our starting point is Aaronson's result that PDQP/qpoly = ALL [6] where he showed how to evaluate π at one point but this assumed that the verifier is provided with a *trusted* quantum advice state. Hence, our contribution is to show that one can retrieve *two* points of choice even if the quantum proof is from an *untrusted prover*.

▶ **Theorem 5.** PDQMA = NEXP.

Proof. PDQMA $\subseteq$ NEXP is clear, since in NEXP we can guess an exponentially-long classical description of the PDQMA witness and then verify it.

Thus, we show NEXP $\subseteq$ PDQMA. By Theorem 4, it suffices to show $\mathsf{PCP}_{c,s}[\mathrm{poly}(n), 2]_\Sigma \subseteq$ PDQMA for some constant-size alphabet Σ. In particular, we can assume the PCP verifier makes two queries to a PCP and receives responses in a constant-size alphabet. Let $\pi : \{0,1\}^n \to \Sigma$ be the Boolean function that encodes the PCP for all possible queries $x \in \{0,1\}^n$. Let $p : \mathbb{F}_q^n \to \mathbb{F}_q$ be the unique degree-n multilinear extension of π, where q is chosen so that the conditions of Lemma 1 are satisfied and $q \geq n+2$. Note that $q = O(n)$ by Bertrand's postulate. To simulate the PCP system, it suffices for the verifier to learn $\pi(w) = p(w)$ and $\pi(w') = p(w')$ at two points w and w' of the verifier's choosing, given a witness state $|\psi\rangle$ sent by Merlin.

We explain the verification procedure as we analyze the honest case (i.e., the case when there exists a π such that the PCP verifier accepts with probability at least c). Let $a, b \in \{0,1\}^n$ be distinct points and let $c \in \mathbb{F}_q^n$ be independent of a and b. Let A be the unique affine plane that contains a, b, and c. Let $C_{a,b,c}$ be the following function. For a vector $y \in \mathbb{F}_q^n$, if $y \in A$, then $C_{a,b,c}(y) = 0^n$. Otherwise, a, b, c, and y define a unique affine cube, which we denote by B. In this case, $C_{a,b,c}(y) = y' \in \mathbb{F}_q^n$ is a canonical representation of the point y, so that (a, b, c, y) and (a, b, c, y') define the same affine cube.

We describe the canonical representation in more detail. First, note that y must be one of the $q^3 - q^2$ points in the cube B that are not in the plane A (otherwise $C_{a,b,c}(y) = 0^n$). There are many ways to pick a canonical representative y'. For example, of the $q^3 - q^2$ many points, one can have $C_{a,b,c}(y)$ output the point y' with the fewest nonzero entries (and if there are ties, pick the one that comes first in lexicographic order). This ensures that all of the $q^3 - q^2$ points get mapped to the same canonical representative y', which is crucial for our verification procedure. Recall that $q = O(n)$, so $C_{a,b,c}$ can be computed efficiently.

The honest PDQMA witness is

$$\frac{1}{\sqrt{q^n}} \sum_{z \in \mathbb{F}_q^n} |z\rangle |p(z)\rangle.$$

Given this witness, the PDQP verification procedure is as follows:

(1) Simulating the PCP verifier, choose two queries $w, w' \in \{0,1\}^n$. Pick a point $w'' \in \mathbb{F}_q^n$ uniformly at random.

(2) Map the witness to

$$\frac{1}{\sqrt{q^n}} \sum_{z \in \mathbb{F}_q^n} |z\rangle |p(z)\rangle |C_{w,w',w''}(z)\rangle.$$

(3) Measure the $|C_{w,w',w''}(z)\rangle$ register in the usual collapsing way to obtain the outcome $y \in \mathbb{F}_q^n$. If the measurement outcome is 0^n, reject. Let B denote the affine cube containing w, w', w'', and y, and let A denote the affine plane containing w, w', and w''.

(4) Make $O(n^4)$ non-collapsing measurements of the $|z\rangle$ and $|p(z)\rangle$ registers.
(5) If exactly the $q^3 - q^2$ points in $B \setminus A$ are obtained and the empirical distribution over these points is $O(1/n)$-close in total variation distance to the uniform distribution, continue. Otherwise, reject.
(6) If more than one $p(z)$ value was obtained for the same z, reject.
(7) Perform polynomial interpolation to obtain trivariate polynomial $u_B : \mathbb{F}_q^3 \to \mathbb{F}_q$ of degree at most n that is consistent with p on the measured points. If this interpolation fails, then reject.
(8) Calculate $\pi(w) = p(w) = u(0,0,0)$ and $\pi(w') = p(w') = u(0,1,0)$. Plug these responses into the PCP verifier, and accept if and only if it does.

Let us analyze the verification procedure in more detail. Suppose, upon measuring the register $|C_{w,w',w''}(z)\rangle$ in Step 3, the verifier sees $y \in \mathbb{F}_q^n$. With probability q^{2-n}, we observe $y = 0^n$, because there are q^2 points on the affine plane $\{w+(w'-w)t_1+(w''-w)t_2\}_{t_1,t_2\in\mathbb{F}_q}$. In this case, the verifier will reject (which is incorrect). Otherwise, with probability $1 - q^{2-n} = 1 - \exp(-\Omega(n))$, the verifier will see some canonical point $y \in \mathbb{F}_q^n$ so that the points (w, w', w'', y) defines an affine cube B.

The post-measurement state of the remaining two registers will then be in superposition over all points $z \in \mathbb{F}_q^n$ such that z, w, w', and w'' define the same affine cube as y, w, w', and w''. Recall that the affine cube B is the set

$$B = \{w + (y-w)t_1 + (w'-w)t_2 + (w''-w)t_3\}_{t_1,t_2,t_3\in\mathbb{F}_q},$$

and the affine plane $A \subseteq B$ containing w, w', and w'' is the set

$$A = \{w + (w'-w)t_1 + (w''-w)t_2\}_{t_1,t_2\in\mathbb{F}_q}.$$

Observe that z can be any of the $q^3 - q^2$ points in $B \setminus A$. In particular,

$$z = w + (y-w)t_1 + (w'-w)t_2 + (w''-w)t_3$$

for any $t_1 \in \mathbb{F}_q^*$ and $t_2, t_3 \in \mathbb{F}_q$. Therefore, our post-measurement state $|\phi\rangle$ can be expressed as

$$\frac{1}{q\sqrt{q-1}} \sum_{\substack{t_1\in\mathbb{F}_q^* \\ t_2\in\mathbb{F}_q \\ t_3\in\mathbb{F}_q}} |w+(y-w)t_1+(w'-w)t_2+(w''-w)t_3\rangle|p(w+(y-w')t_1+(w'-w)t_2+(w''-w)t_3)\rangle.$$

Define $u_B : \mathbb{F}_q^3 \to \mathbb{F}$ by $u(t_1,t_2,t_3) \coloneqq p(w + (y-w)t_1 + (w'-w)t_2 + (w''-w)t_3)$, and notice that $u_B(0,0,0) = p(w)$ and $u_B(0,1,0) = p(w')$. Because p is the multilinear extension of π, we also have that $p(w) = \pi(w)$ and $p(w') = \pi(w')$.

After Step 6, the verifier has collected $q^3 - q^2$ pairs $(z, p(z))_{z\in B\setminus A}$. Collecting these pairs is an instance of the coupon collector's problem, so $O(q^3 \log q) = O(n^3 \log n)$ many samples suffice to succeed with high probability. We take more samples, which will be relevant to the soundness of our protocol. With the $(z, p(z))$ pairs, the verifier runs polynomial interpolation to learn the polynomial u_B. We note that q is chosen to be $\geq n + 2$ to ensure that $q^3 - q^2$ pairs suffice for polynomial interpolation. After learning u_B, the verifier has learned $\pi(w)$ and $\pi(w')$ as desired and will accept with the same probability as the PCP verifier.

A crucial part of our verification procedure is that the verifier tests that p is a low-degree polynomial. Because we picked a point $w'' \in \mathbb{F}_q^n$ uniformly at random, we ensure that the affine cube B contains a random affine line, independent of the points w and w'. Therefore,

the polynomial interpolation succeeds if and only if the truth table of p matches a low-degree polynomial on a randomly chosen line ℓ. Hence, by Lemma 1, p must also be globally low degree. After taking into account the failures that can occur during the verification procedure, we conclude that the verifier accepts with probability at least $c - \exp(-\Omega(n))$.

We now analyze the soundness of our verification procedure. That is, suppose the PCP verifier will accept with probability at most s for all possible proofs π. We will show that the PDQMA verifier accepts with probability at most $s + \exp(-\Omega(n))$. The key insight for the soundness case is that, by deviating from the honest witness state above, Merlin only increases the probability that the polynomial interpolation will fail, causing Arthur to reject. In particular, the only way Merlin can cheat is to encode a truth table in $|\psi\rangle$ that is not degree n, but some function with much larger degree. However, the verifier will detect this with the lines-point low-degree test.

Before going through the technical details, let us emphasize that Merlin does not know the points w, w', and w'' that the verifier will select – these are chosen after Merlin commits to a witness state. Hence, Merlin must send a witness state $|\psi\rangle$ that passes all the checks in the verification procedure for all choices of w, w', and w'' and no matter the random outcome y the verifier observes in Step 3.

Formally, Merlin can send an arbitrary state:

$$|\psi\rangle = \sum_{z \in \mathbb{F}_q^n, b \in \mathbb{F}_q} \alpha_{z,b} |z\rangle|b\rangle.$$

The verifier maps the witness to

$$\sum_{z \in \mathbb{F}_q^n, b \in \mathbb{F}_q} \alpha_{z,b} |z\rangle|b\rangle|C_{w,w',w''}(z)\rangle,$$

and measures the last register. Suppose the measurement outcome is some $y \in \mathbb{F}_q^n$. If $y = 0^n$, the verifier rejects, but we will pessimistically assume this never happens. Suppose $y \neq 0^n$. As discussed previously, there are $q^3 - q^2$ points z such that z, w, w', and w'' define the same affine cube B as y, w, w', and w''. These correspond to the points in $B \setminus A$ (recall that A is the affine plane containing w, w', and w''). Define $D \subseteq B \setminus A$ to be the points $z \in B \setminus A$ for which there exists at least one nonzero $\alpha_{z,b}$ for some $b \in \mathbb{F}_q$. The post-measurement state $|\phi\rangle$ is then

$$|\phi\rangle = \sum_{z \in D, b \in \mathbb{F}_q} \widetilde{\alpha_{z,b}} |z\rangle|b\rangle,$$

where

$$\widetilde{\alpha_{z,b}} = \frac{|\alpha_{z,b}|}{\sqrt{\sum_{z \in D, b \in \mathbb{F}_q} |\alpha_{z,b}|^2}}.$$

Note that we can assume without loss of generality that $\widetilde{\alpha_{z,b}} \in \mathbb{R}$ as the verifier only performs non-collapsing measurements on $|\phi\rangle$. In fact, in Step 4, the verifier's actions can be understood as drawing samples from a classical probability distribution where each (z, b) pair has a probability of $|\widetilde{\alpha_{z,b}}|^2$.

Recall the well-known fact that $\Theta(\frac{n + \log(1/\delta)}{\varepsilon^2})$ samples are necessary and sufficient to learn a distribution to total variation distance at most ε with probability at least $1 - \delta$ (cf. [18, Theorem 1]). Hence, the $O(n^4)$ non-collapsing measurements in Step 5 suffice to learn the distribution over (z, b) pairs in the support of $|\phi\rangle$ to total variation distance at most

$O(1/n)$ with probability at least $1 - \exp(-\Omega(n))$. In particular, for Steps 4 through 6 to pass, $|\phi\rangle$ must be (approximately) uniformly supported on pairs (z, b_z) for each $z \in B \setminus A$. (The verifier will immediately reject if any $z \in B \setminus A$ is paired with more than one $b \in \mathbb{F}_q$ value.) Because this must hold for any affine plane A and affine cube B, we can deduce that Merlin is forced to send a state that is (approximately) uniform over pairs (z, b_z) for every $z \in \mathbb{F}_q^n$.

Assuming these steps pass, Step 7 passes only if the observed pairs $(z, b_z)_{z \in B \setminus A}$ fit a degree-n trivariate polynomial u_B. As discussed in the honest case, by selecting a $w'' \in \mathbb{F}_q^n$ uniformly at random, we guarantee that the affine cube B contains a random line ℓ, independent of the w and w' (this is precisely why we need a 3-dimensional affine subspace). Therefore, the interpolation succeeding implies that the function encoded by Merlin matches a degree-n polynomial on a randomly chosen line. By Lemma 1, we can conclude that Merlin indeed encoded a truth table for a degree-n polynomial.

If all of these steps pass, then the verifier can learn $\pi(w)$ and $\pi(w')$ as desired by evaluating $u_B(0,0,0)$ and $u_B(0,1,0)$. By plugging the values $\pi(w)$ and $\pi(w')$ into the PCP verifier, the PDQMA verifier accepts with probability at most s. ◀

4 QMA and Hidden Variables

We introduce and characterize the complexity class DQMA, a variant of QMA where the verifier can perform DQP computations. Informally, DQP is like BQP with the ability to inspect the entire history of a hidden variable. For completeness, we begin this section by giving a formal definition of DQMA. Then, as with PDQMA, we prove that DQMA = NEXP. This result is more "quantum" than PDQMA = NEXP (Theorem 5) because the DQP verifier will use quantum circuits (as opposed to merely computational basis measurements).

4.1 The Complexity Class DQMA

We give a formal definition of DQMA. To do so, we must recall a few definitions related to the class DQP (see [4] for more detail about this class).

We begin by defining hidden-variable theories. To aid intuition, one can think of hidden-variable theories like standard quantum mechanics where there is a state vector that evolves unitarily. However, there is also a deeper "hidden variable" in some definite state (i.e., not in superposition) that evolves stochastically in a manner determined by the state vector and the state vector's unitary evolution. In the series of definitions that follow, we are building up to defining a model of computation where one evolves a state vector unitarily, and then (at the end of the computation) can inspect which states the hidden variable was in at each step of the computation.

▶ **Definition 6** (Hidden-variable theory). *A hidden-variable theory is a family of functions $\{S_d\}_{d \in \mathbb{N}}$, where each S_d maps a d-dimensional mixed state ρ and a $d \times d$ unitary matrix U onto a singly stochastic matrix $S_d(\rho, U)$.*

That is, we take a hidden-variable theory to be a function that maps the unitary evolution of a state to a stochastic matrix that evolves one probability distribution to another. Conditioned on a hidden variable being in a state $|j\rangle$, $(S)_{ij}$ is the probability that a hidden variable transitions to the $|j\rangle$.

Aaronson [4] defined a number of axioms a hidden-variable theory could satisfy. We require three of these axioms to define DQP: the marginalization axiom, the indifference axiom, and the robustness axiom. In the following definitions, S denotes the hidden-variable theory, ρ a d-dimensional quantum state, and U a $d \times d$ unitary matrix. Each axiom must hold for all $d \in \mathbb{N}$.

▶ **Definition 7** (Marginalization axiom). *The marginalization axiom says that for all* $j \in \{1, \ldots, d\}$,

$$\sum_i (S)_{ij}(\rho)_{ii} = (U\rho U^\dagger)_{jj}.$$

In words, the marginalization axiom says that the hidden-variable theory should make predictions that are consistent with quantum mechanics.

▶ **Definition 8** (Indifference axiom). *For a matrix* $M \in \mathbb{C}^{d\times d}$, *let a block be a subset* $B \subseteq \{1, \ldots, d\}$ *such that* $(M)_{ij} = 0$ *for all* (i, j) *such that* $i \in B$, $j \notin B$ *and* $i \notin B$, $j \in B$. *The indifference axiom says that the stochastic matrix* $S(\rho, U)$ *must have the same blocks as* U.

Physically, the indifference axiom says the following. Given any quantum state ρ in a tensor product space $\mathcal{H}_A \otimes \mathcal{H}_B$ and any unitary U acting nontrivially only on $\mathcal{H}_A$, the stochastic matrix $S(\rho, U)$ acts nontrivially only on $\mathcal{H}_A$ as well.

Finally, we state the robustness axiom, for which we need the following notation. Let $P(\rho, U)$ be the matrix of joint probabilities whose (i, j) entry is $(P)_{ij} \coloneqq (S)_{ij}(\rho)_{ii}$.

▶ **Definition 9** (Robustness axiom). *Let* $\widetilde{\rho}$ *and* $\widetilde{U}$ *be perturbations of* ρ *and* U, *respectively, and, for a matrix* M, *let* $\|M\|_\infty \coloneqq \max_{i,j} |(M)_{ij}|$. *The robustness axiom says that, for all polynomials* p, *there should exist a polynomial* q *such that,*

$$\|P(\widetilde{\rho}, \widetilde{U}) - P(\rho, U)\|_\infty \leq \frac{1}{p(d)},$$

whenever $\|\widetilde{\rho} - \rho\|_\infty \leq 1/q(d)$ *and* $\|\widetilde{U} - U\| \leq 1/q(d)$.

The robustness axiom is necessary to prove that the class DQP is not sensitive to the choice of gate set defining the class.

Next, we define the history of a hidden variable.

▶ **Definition 10** (Hidden variable history). *Let* $|\psi_{\text{init}}\rangle$ *be an* n*-qubit quantum state, and let* $U \coloneqq U_T \cdots U_1$ *be an* n*-qubit, depth-*T *quantum circuit, where* $U_1, \ldots, U_T$ *denote each layer of the quantum circuit* U. *The history of a hidden variable is a sequence* $H = (v_0, \ldots, v_T)$ *of basis states, where* v_t *is the state of the hidden variable immediately after the layer* U_t *of the circuit is applied. Given a hidden-variable theory* $\mathcal{T} \coloneqq \{S_d\}_{d\in\mathbb{N}}$, *we obtain a probability distribution over hidden variable histories* $\Omega(\mathcal{T}, U, |\psi_{\text{init}}\rangle)$ *via the stochastic matrices*

$$S(|\psi_{\text{init}}\rangle, U_1),\ S(U_1|\psi_{\text{init}}\rangle, U_2), \ldots, S(U_{T-1}\cdots U_1|\psi_{\text{init}}\rangle, U_T).$$

We can now define the complexity class DQMA.

▶ **Definition 11.** DQMA(c, s) *is the class of languages* $L \subseteq \{0,1\}^*$ *for which there exists a* DQP *verifier* V *(to be defined shortly) such that, for all* $x \in \{0,1\}^*$*:*

- ***Completeness:*** *If* $x \in L$ *then there exists a witness state* $|\phi\rangle$, *on* poly(n) *qubits, such that* $V(x, |\phi\rangle)$ *accepts with probability at least* c.
- ***Soundness:*** *If* $x \notin L$ *then* $V(x, |\phi\rangle)$ *accepts with probability at most* s *for all witness states* $|\phi\rangle$.

A DQP *verifier is defined as follows. Let* $\mathcal{T}$ *be a hidden-variable theory satisfying the marginalization, indifference, and robustness axioms (Definitions 7–9), and let* C_x *(depending on the input* x*) be a* P*-uniform quantum circuit comprised of gates from any finite gate set that is universal for* BQP. *A* DQP *verifier is a deterministic classical Turing machine that is allowed to draw one sample from the distribution* $\Omega\left(\mathcal{T}, C_x, |\phi\rangle|0^{p(n)}\rangle\right)$ *(Definition 10), where* $|\phi\rangle$ *is the witness and* p *is a polynomial.*

4.2 DQMA = NEXP and DQP/qpoly = ALL

We conclude this section by proving DQMA = NEXP and DQP/qpoly = ALL. Recall that in the proof of PDQMA = NEXP (Theorem 5), the verifier uses non-collapsing measurements to sample $O(n \log n)$ values of a multivariate polynomial along an affine line of one's choice and then does interpolation. To prove DQMA = NEXP, we use the same verification procedure, except we use the history of a hidden variable to collect the samples in place of non-collapsing measurements. Hence, to prove DQMA = NEXP it suffices to explain how we collect these samples with the history of a hidden variable. We achieve this by generalizing the "juggle subroutine" due to Aaronson [4, Section VII].

▶ **Lemma 12** (Juggle subroutine [4, Section VII]). *Suppose we have an ℓ-qubit state*

$$\frac{|a\rangle \pm |b\rangle}{\sqrt{2}},$$

where $|a\rangle$ and $|b\rangle$ are unknown basis states. Given a single copy of the state, the juggle subroutine (a DQP *algorithm) can efficiently learn both a and b with success probability at least $1 - e^{-\ell}$.*

We generalize this algorithm to work on states that are an equal superposition over polynomially many strings by reducing to the case of an equal superposition over two strings. Before explaining the generalization, we first must define (and give a simple fact about) pairwise independent families of hash functions, which are used in our reduction.

▶ **Definition 13** (Pairwise independent family of hash functions). *A family of hash functions $\mathcal{H} = \{h : \{0,1\}^\ell \to R\}$ is called pairwise independent if $\forall\ x \neq y \in \{0,1\}^\ell$ and $\forall a_1, a_2 \in R$, we have*

$$\Pr[h(x) = a_1 \wedge h(y) = a_2] = \frac{1}{|R|^2}.$$

▷ Claim 14. Let $S \subseteq \{0,1\}^\ell$ be a subset, and let $\mathcal{H} \coloneqq \{h : \{0,1\}^\ell \to R\}$ be a family of pairwise independent hash functions such that $2|S| - 4 \leq |R| \leq 3|S| - 3$. Then, for any fixed $x \in S$, the probability that x collides with exactly one other $y \in S$ is at least $\frac{1}{6}$.

Proof. Let $h \in \mathcal{H}$ be chosen uniformly at random, and let $x \in S$ be some fixed element. The probability that exactly one other element collides with x is

$$\frac{|S| - 1}{|R|} \cdot \left(1 - \frac{1}{|R|}\right)^{|S|-2}.$$

We lower bound this quantity.

$$\begin{aligned}
\frac{|S| - 1}{|R|} \cdot \left(1 - \frac{1}{|R|}\right)^{|S|-2} &\geq \frac{|S| - 1}{|R|} \cdot \left(1 - \frac{|S| - 2}{|R|}\right) \\
&\geq \frac{1}{3} \cdot \left(1 - \frac{|S| - 2}{|R|}\right) \\
&\geq \frac{1}{6}.
\end{aligned}$$

The first inequality follows from Bernoulli's inequality. The second and third inequalities use the fact that $2|S| - 4 \leq |R| \leq 3|S| - 3$. ◁

We now give the generalized juggle subroutine.

▶ **Lemma 15** (Generalized juggle subroutine). *Suppose we have an ℓ-qubit state*

$$\frac{1}{\sqrt{|S|}}\sum_{x\in S}|x\rangle,$$

where $S \subseteq \{0,1\}^\ell$ is an unknown subset of $|S| \leq \mathrm{poly}(\ell)$ basis states. Given a single copy of the state, the generalized juggle subroutine (a DQP *algorithm) can efficiently learn S with success probability at least $1 - e^{-\ell}$.*

Proof. We begin with the state of the form

$$|\psi\rangle = \frac{1}{\sqrt{|S|}}\sum_{x\in S}|x\rangle.$$

We perform a procedure that involves applying transformations to $|\psi\rangle$ and then inverting them to get back to $|\psi\rangle$, in an attempt to "dislodge" the hidden variable from whichever basis state it's currently sitting in, and get it into a different, uniformly random one. Each time we do this, we have a $1/\mathrm{poly}(\ell)$ probability of success. Importantly, since there's no penalty for failure, we can repeat this procedure $\mathrm{poly}(\ell)$ times, and then with overwhelming probability, the hidden variable will have visited every basis state $|x\rangle$ for $x \in S$. Therefore, one will learn S upon observing the hidden-variable history.

The procedure works as follows. First, choose a random hash function h (with a range satisfying the conditions in Claim 14) from some pairwise-independent family, and then map each $|x\rangle$ to $|x\rangle|h(x)\rangle$. Let y be the current state of the hidden variable. By Claim 14, with probability at least $\frac{1}{6}$, there's *exactly* one other basis state $z \neq y$ such that $h(z) = h(y)$, and this z is uniformly random. If that happens, then because of the indifference axiom (which says that, if we don't touch the h-register, then the hidden variable will never move between h-values), we've reduced to the problem handled by the original juggle subroutine. In particular, we can now run the original juggle subroutine on the first register (where the hidden variable is). Lemma 12 tells us that, with probability at least $1 - e^{-\ell}$, the hidden variable moves from y to z. Finally, we uncompute h, leaving us with our original state $|\psi\rangle$. Overall, the inner loop moves the hidden variable from y to a uniformly random z with probability at least $\frac{1}{6} - \frac{e^{-\ell}}{6} \geq 1/\mathrm{poly}(\ell)$.[3] Since there is no penalty for failure, we can repeat this procedure $2\ell^2$ times to ensure that with probability at least $1 - e^{-\ell}$, the hidden variable was successfully moved to a uniformly random basis state. Finally, since we visit a uniformly random state with high probability, we can visit every state with high probability by repeating this entire procedure a polynomial number of times. ◀

We are now ready to prove the main theorem of this section. Namely, that giving Arthur access to hidden-variable histories blows up the power of QMA to NEXP.

▶ **Theorem 16.** DQMA = NEXP.

Proof. It is clear that DQMA $\subseteq$ NEXP. By Theorem 4, we can complete the proof by showing MIP $\subseteq$ DQMA.

The verification procedure and the honest witness sent by Merlin are the same as in the proof of Theorem 5. Suppose we have a yes-instance, and Merlin sends the honest witness with the form:

$$\frac{1}{q^n}\sum_{z\in\mathbb{F}_q^n}|z\rangle|p(z)\rangle.$$

[3] We note that there is some chance that the hidden variable stays put or moves to somewhere other than z, but that's OK too, since our ultimate goal is for the hidden variable to visit every possible state in S. In any case, we keep repeating.

We must explain how to use the history of a hidden variable in lieu of non-collapsing measurements. After the verifier makes *collapsing* measurements, they must learn the support of the post-measurement state. To do this, the verifier runs the generalized juggle subroutine (Lemma 15). After that, the verifier can do polynomial interpolation with just classical computation (or reject if there is insufficient data to do the interpolation).

The only difference in the verification procedure is that the data for polynomial interpolation is collected via the generalized juggle subroutine instead of non-collapsing measurements. Therefore, the completeness and soundness of this procedure follow in the same way as for Theorem 5. In particular, if Merlin deviates from the honest witness state, then he can only hurt his success probability by causing the polynomial interpolation step to fail. ◀

Finally, we remark that our generalized juggle subroutine can be used to prove $\mathsf{DQP/qpoly} = \mathsf{ALL}$, complementing Aaronson's result that $\mathsf{PDQP/qpoly} = \mathsf{ALL}$ [6] and Raz's result that $\mathsf{QIP}(2)/\mathsf{qpoly} = \mathsf{ALL}$ [24]. Similar to Theorem 16, this result is more "quantum" than Aaronson's or Raz's, because the generalized juggle subroutine requires quantum computation.

▶ **Theorem 17.** $\mathsf{DQP/qpoly} = \mathsf{ALL}$.

Proof. The advice state and verification procedure are the same as in [6, Theorem 2], except we replace the non-collapsing measurements with the generalized juggle subroutine in the same manner described in the proof of Theorem 16. ◀

5 Open Problems

Is $\mathsf{PDQP} \subseteq \mathsf{DQP}$? Aaronson et al. [8] give intuition for why this containment ought to be true, but it remains an open problem. Proving $\mathsf{PDQP} \subseteq \mathsf{DQP}$ (combined with Theorem 5) immediately implies $\mathsf{DQMA} = \mathsf{NEXP}$, simplifying our proof in Theorem 16. We also remark that improving the upper bound $\mathsf{DQP} \subseteq \mathsf{EXP}$ remains an interesting open problem.

We now know that modifying QMA by giving the verifier access to non-collapsing measurements, hidden-variable histories, or non-negative witnesses will cause QMA to "explode" in power to NEXP. Is it possible to replace the verifier with some variant that does *not* lead to NEXP? Having access to such variants may find applications in proving better bounds on $\mathsf{QMA}(2)$.

References

1 S. Aaronson. Quantum Lower Bound for the Collision Problem. In *Proc. ACM STOC*, pages 635–642, 2002. `quant-ph/0111102`.

2 S. Aaronson. Is Quantum Mechanics An Island In Theoryspace? In A. Khrennikov, editor, *Proceedings of the Växjö Conference "Quantum Theory: Reconsideration of Foundations"*, 2004. `quant-ph/0401062`.

3 S. Aaronson. Limitations of Quantum Advice and One-Way Communication. *Theory of Computing*, 1:1–28, 2005. Earlier version in CCC'2004. `quant-ph/0402095`. `doi:10.4086/TOC.2005.V001A001`.

4 S. Aaronson. Quantum Computing and Hidden Variables. *Phys. Rev. A*, 71(032325), 2005. `quant-ph/0408035` and `quant-ph/0408119`.

5 S. Aaronson. Quantum Computing, Postselection, and Probabilistic Polynomial-Time. *Proc. Roy. Soc. London*, A461(2063):3473–3482, 2005. `quant-ph/0412187`.

6 S. Aaronson. $\mathsf{PDQP/qpoly} = \mathsf{ALL}$. `arXiv:1805.08577`, 2018. `arXiv:1805.08577`.

7 S. Aaronson, S. Beigi, A. Drucker, B. Fefferman, and P. Shor. The power of unentanglement. In *Proc. Conference on Computational Complexity*, pages 223–236, 2008. arXiv:0804.0802.

8 S. Aaronson, A. Bouland, J. Fitzsimons, and M. Lee. The space "just above" BQP. In *Proc. Innovations in Theoretical Computer Science (ITCS)*, pages 271–280, 2016. arXiv:1412.6507.

9 S. Aaronson and A. Drucker. A Full Characterization of Quantum Advice. *SIAM J. Comput.*, 43(3):1131–1183, 2014. Earlier version in STOC'2010. arXiv:1004.0377.

10 S. Aaronson and J. Watrous. Closed Timelike Curves Make Quantum and Classical Computing Equivalent. *Proc. Roy. Soc. London*, A465:631–647, 2009. arXiv:0808.2669.

11 D. S. Abrams and S. Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. *Phys. Rev. Lett.*, 81:3992–3995, 1998. quant-ph/9801041.

12 D. Aharonov and O. Regev. A Lattice Problem in Quantum NP. In *Proc. IEEE FOCS*, pages 210–219, 2003. quant-ph/0307220.

13 S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *J. of the ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.

14 S. Arora and S. Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *Journal of the ACM (JACM)*, 45(1):70–122, 1998. doi:10.1145/273865.273901.

15 J. Barrett, N. de Beaudrap, M. J. Hoban, and C. M. Lee. The computational landscape of general physical theories. *npj Quantum Information*, 5(1):41, 2019. doi:10.1038/s41534-019-0156-9.

16 R. Bassirian, B. Fefferman, and K. Marwaha. Quantum merlin-arthur and proofs without relative phase, 2023. arXiv:2306.13247.

17 R. Bassirian and K. Marwaha. Superposition detection and QMA with non-collapsing measurements, 2024. arXiv:2403.02532.

18 C. L. Canonne. A short note on learning discrete distributions, 2020. arXiv:2002.11457.

19 D. Dieks. Modal interpretation of quantum mechanics, measurements, and macroscopic behaviour. *Phys. Rev. A*, 49:2290–2300, 1994.

20 K. Friedl and M. Sudan. Some improvements to total degree tests. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pages 190–198. IEEE, 1995. doi:10.1109/ISTCS.1995.377032.

21 P. Harsha, M. Kumar, R. Saptharishi, and M. Sudan. An improved line-point low-degree test, 2023. arXiv:2311.12752.

22 F. G. Jeronimo and P. Wu. The Power of Unentangled Quantum Proofs with Non-Negative Amplitudes. In *Proc. ACM STOC*, pages 1629–1642, 2023. doi:10.1145/3564246.3585248.

23 C. M. Lee and J. Barrett. Computation in generalised probabilisitic theories. *New Journal of Physics*, 17(8):083001, 2015. doi:10.1088/1367-2630/17/8/083001.

24 R. Raz. Quantum Information and the PCP Theorem. *Algorithmica*, 55(3):462–489, 2009. Earlier version in FOCS'2005. quant-ph/0504075. doi:10.1007/S00453-007-9033-6.

25 T. Rudolph and L. Grover. A 2 rebit gate universal for quantum computing, 2002. arXiv:quant-ph/0210187.

26 E. Schrödinger. Über die Umkehrung der Naturgesetze. *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-Mathematische Klasse*, 8(9):144–153, 1931. English title: On the Reversal of the Laws of Nature.

How Pinball Wizards Simulate a Turing Machine

Rosemary U. Adejoh
Faculty of Media, Bauhaus-Universität Weimar, Germany

Andreas Jakoby
Faculty of Media, Bauhaus-Universität Weimar, Germany

Sneha Mohanty
Computer Networks and Telematics, University of Freiburg, Germany

Christian Schindelhauer
Computer Networks and Telematics, University of Freiburg, Germany

Abstract

We introduce and investigate the computational complexity of a novel physical problem known as the *Pinball Wizard* problem. It involves an idealized pinball moving through a maze composed of one-way gates (outswing doors), plane walls, parabolic walls, moving plane walls, and bumpers that cause acceleration or deceleration. Given the initial position and velocity of the pinball, the task is to decide whether it will hit a specified target point.

By simulating a two-stack pushdown automaton, we show that the problem is Turing-complete – even in two-dimensional space. In our construction, each step of the automaton corresponds to a constant number of reflections. Thus, deciding the *Pinball Wizard* problem is at least as hard as the Halting problem. Furthermore, our construction allows bumpers to be replaced with moving walls. In this case, even a ball moving at constant speed – a so-called ray particle – can be used, demonstrating that the *Ray Particle Tracing* problem is also Turing-complete.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Pinball Wizard problem, Halting problem, Turing-complete

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.4

Related Version *Full Version*: `https://arxiv.org/abs/2510.02560` [2]

1 Introduction and Motivation

A question of continuous interest is which physical or analog models are Turing complete. Christopher Moore [16] outlined a construction showing that, even with precisely known initial conditions, friction less movement, perfect reflection and arbitrary precision, a single billiard ball moving at constant speed can simulate a two-counter machine in 3D. Similarly, Reif et al. [20] demonstrated in their seminal work that 3D ray tracing can simulate a reversible two-counter automaton, thereby establishing its Turing-completeness. Both models rely on the use of parabolic mirrors in 3D space, supporting Moore's conjecture that three dimensions suffice to achieve Turing universality.

In this paper, we show that full 3D Euclidean space is not required: by allowing the speed of the ball to vary, we introduce a novel two-dimensional model – referred to as the 2D Pinball Wizard Problem, that can simulate a Turing complete two-stack pushdown automaton (PDA) [12].

Our *Pinball Wizard problem* is inspired by classical pinball machines and features components such as reflecting planes and parabolic walls, one-way gates (outswing doors), bumpers for accelerating and decelerating the ball, as well as a moving plane wall. Unlike in real pinball machines we do not have a tilted plane and the ball, described by a single point, moves in a straight line with constant velocity when it does not bounce into one of the components.


© Rosemary U. Adejoh, Andreas Jakoby, Sneha Mohanty, and Christian Schindelhauer;
licensed under Creative Commons License CC-BY 4.0

These components may change the direction of the ball's trajectory and the velocity of the pinball when the pinball bounces on these components in its path. See Section 3 for a complete definition of this problem.

The main components of the original pinball game that we retain here, are the one-way gates as well as the target. A one-way gate is an area that is blocked off after the ball passes through it once. It does not change the movement of a ball coming from one side, but will block and reflect the ball if it comes from the other side. We also use bumper walls in our work, which when hit by the pinball, pushes the ball away with a positive or negative change of speed. While there are various types of target in the pinball game, for simplicity, we are working with stationary target position.

The 2D Pinball Wizard model describes a decision problem, where given an initial configuration of the pinball, it is to be decided, whether it hits a given target. We show that this model facilitates the simulation of any two-stack automaton with push and pop operations.

If all stationary bumpers are replaced with a combination of moving walls, the result still holds. We consider the spatial offset interval where the ball enters the construction, and the modified speed determines the time delay (offset) at the end line. These components serve our design of a pinball machine where we want to test if the ball exits at a certain target position given a start time and space offset at the beginning of the path of a pinball having a known speed and direction. Finally, when we exchange the ball with a ray particle with constant speed, a modified construction holds up as well, showing that the *Ray Particle Tracing* problem with moving walls and one-way mirrors is also Turing-complete.

In Section 2, we discuss Related Work. Thereafter, we provide a more formal definition of the *Pinball Wizard* problem in Section 3. That Section concludes with a brief introduction of the *Ray Particle Tracing* problem, and its relationship with our *Pinball Wizard* problem. We then establish the Turing-completeness of the *Pinball Wizard* problem by simulating a two-stack pushdown automaton in Section 4. Subsequently, we modify our construction to briefly demonstrate analogous results for the *Ray Particle Tracing* problem in Section 5. Finally, Conclusions and Open Problems are presented in Section 6.

2 Related Work

The computational complexity of puzzles and games has long fascinated mathematicians and computer scientists. In his survey [21], Uehara outlines the history of this line of research and shows current trends in theoretical computer science. The simple rules found in puzzles and games not only resemble basic computational operations, but can also serve as alternative models of computation. Studying them may lead to deeper insights into the nature of computation itself.

A remarkable result in this context is the Game of Life, popularized by Martin Gardner in [11]. Confirming a long-standing conjecture, Berlekamp et al. [3] showed in 2004 that it is Turing-complete – that is, any computation performed by a Turing machine can be simulated within Conway's Game of Life.

Sometimes, games are invented to gain a better understanding of aspects of computation. In 1974, Cook and Reckhow [6] introduced the pebble game to study storage requirements. This game has proven to be a powerful tool for exploring computational complexity classes such as NL, P, NP, PSPACE, EXP, and others. It also plays an important role in understanding register allocation, parallel computation, and proof complexity.

There is not enough room to cover all the research on the computational complexity of traditional games and puzzles published since the advent of NP-completeness and higher complexity classes such as PSPACE. For example, Demaine et al. in [7] prove that push-pull block puzzles in 3D are PSPACE-complete to solve, and that push-pull block puzzles in 2D with thin walls are NP-hard, thereby settling an open question posed in [24]. Recent progress in the field of combinatorial reconfiguration, particularly through the use of the constraint logic model, has revealed many natural PSPACE-complete problems. These include a variety of sliding block puzzles whose computational complexity had remained open for nearly 40 years.

For example, in [9], Forišek analyzes the computational complexity of various two-dimensional platform games. He states and proves several meta-theorems that identify classes of games for which the set of solvable levels is NP-hard, and others for which the set is even PSPACE-hard.

Pinball Wizard is a very popular game, often found in pubs as a contrived mechanical device or as a video/computer game. In the realm of video games, speedrunning is a popular activity where the goal is to complete a game as quickly as possible. Well-known titles such as Super Mario Bros., Castlevania, and Mega Man are played by enthusiasts worldwide, with countless hours spent daily on livestreams as players refine their skills in pursuit of world records. However, human execution is not the only factor in a successful speedrun. Common techniques such as damage boosting and routing require careful planning to maximize time gains. In [13], Lafond shows that optimizing these mechanics constitutes a significant algorithmic challenge, leading to novel generalizations of classic NP-hard problems such as the knapsack and feedback arc set problems.

The fascination of Pinball Wizard may lie in its physical nature: a ball moving on a plane, representing an analog computational model not captured by traditional digital models discussed so far. Analog models offer many intriguing facets. In [4], Bonifaci et al. provide a rigorous proof that a biologically inspired mathematical model of slime mold behavior converges to the shortest path in any network, regardless of topology or initial mass distribution – formally validating classical maze experiments. They generalize the original model by Nakagaki et al. [17] to show convergence not only for shortest paths but also for undirected linear programs with non-negative cost vectors.

Analog models can display surprising limitations. Pour-El and Richards [18] construct a computable initial value problem – based on well-behaved differential equations with computable coefficients and initial conditions – whose solution is not computable. This striking result demonstrates that physical systems governed by differential equations may yield unpredictable outcomes, even with perfect knowledge of the inputs.

The reversible nature of physical computation is explored in Fredkin and Toffoli's Billiard Ball Model (BBM) [10], in which multiple balls and elastic collisions – akin to carom billiards – are used to simulate logic gates. Margolus [14] extends this model to show that conservative, reversible systems can simulate universal computation, making outcome prediction undecidable in the general case. For further constructions, see Durand-Lose's textbook [8]. As in our work, precise timing is crucial for correctness. Zhang et al. [22] optimize the mirror-based collision computing model by introducing m-counting gates, significantly reducing the number of mirrors required. A comprehensive overview of the field of collision-based computation can be found in [1]. In [5], Chattopadhyay and Gayen have shown that 3-bit XOR and XNOR logic circuits can be constructed using mechanical elements, such as; beam combiners, fixed as well as movable mirrors.

As mentioned earlier, in 1990 Moore [16] envisioned a simulation in which a single billiard ball moving at constant speed in 3D can simulate a two-counter machine. A more detailed proof can be found in the work of Reif et al. [20], showing that 3D ray tracing (the same problem with a different name) is Turing-complete. Their seminal work analyzes six 3D scenarios and was the first to investigate the computational complexity of ray tracing, establishing it as equivalent to the Halting problem. This foundational result has sparked significant theoretical discussions, such as those in [23]. The intricate nature of Ray Tracing has recently inspired a proposal in [15] to employ optical gates using reflection and refraction as foundational elements in a symmetric encryption algorithm.

We are not the first to consider the theoretical aspects of bumpers in a pinball machine. Pring et al. [19] study the dynamics of an impact oscillator with a modified reset law, inspired by a system with "active impact" – namely, the pinball machine. They show how the subtle interplay between two underlying maps gives rise to a large number of new periodic orbits, potentially explaining some of the complexity observed in the dynamics of real pinball machines.

Moore [16] and Reif et al. [20] demonstrated that three spatial dimensions are sufficient for Turing-completeness, leaving open the question of whether two dimensions would suffice. While our construction still relies on speed or time as a third dimension, we show that the spatial aspect of the system can be confined to two dimensions while preserving Turing-completeness.

3 The Pinball Wizard problem

We introduce the Pinball Wizard problem as follows: The pinball is a point moving in two dimension Euclidean space with variable speed and direction. It moves through a pinball machine and interacts with the components elaborated below under *Components*.

The ball enters the construction at timestep $t = 0$ with a given speed and direction. The task is to determine whether this pinball reaches a given target. We can analyze the movement of the ball within the construction, which consists of stationary and moving walls, one-way gates, parabola walls and bumper walls. With these components we later on construct gadgets implementing two independent stacks using space offset and time delay (offset) while changing the speed on the pinball, when it interacts with our gadgets. Some gadgets, are dedicated to inflict controlled time delays produced by pairwise bumpers or pairwise moving walls.

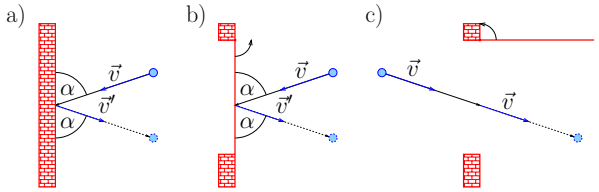


Figure 1 a) A perfectly elastic collision of a ball of speed $\vec{v}$ with a wall, after the collision the direction has changed but the speed is preserved. b) A ball has a collision with a one-way gate on the blocking side. This leads to a perfectly elastic collision. c) A ball has a collision with a one-way gate on the pass through side.

In the Pinball Wizard problem we always assume perfectly elastic collisions of the ball with walls, where the ball might change speed and direction without any impact on the wall.

Versions of perfectly elastic collisions possible for the Pinball Wizard problem

1. The wall is stationary. Here the ball changes its path after hitting the wall according to the law of reflection (see Figure 1.a).
2. The wall is moving with a constant speed. Then we can investigate the system relative to the moving wall which changes the speed and travel direction of the ball. We determine the new speed and direction of the ball relative to the wall's inertia, see Figure 2.
3. The wall is accelerated, i.e. the speed changes over time. Then we analyze the speed of the wall at the time of collision, and determine the new direction of the ball and its speed based on the speed and the direction of the movement of the wall at the time of collision. To determine the new direction and speed of the ball, we apply case 2 above.

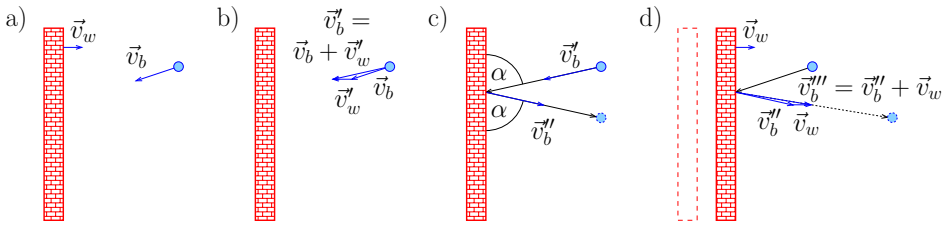


Figure 2 a) A perfectly elastic collision of a ball of speed $\vec{v}_b$ with a wall of speed $\vec{v}_w$ b) relative to the wall the ball has a speed of $\vec{v}_b' = \vec{v}_b + \vec{v}_w'$ c) the collision and the speed of the ball $\vec{v}_b''$ relative to the wall after the collision and d) the balls direction and speed $\vec{v}_b''' = \vec{v}_b'' + \vec{v}_w$ if it is not analyzed relative to the moving wall.

We also use bumper walls, which have a similar effect as straight moving walls. Bumpers increase or decrease the magnitude of the ball's velocity vector while reversing its direction, reflecting the ball to the opposite direction relative to its path before the collision. This behavior is illustrated in Figure 3.

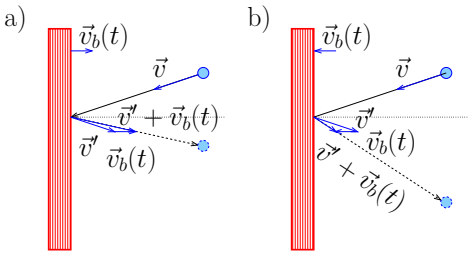


Figure 3 a) A bumper with positive (speed increasing) effect and b) a bumper with negative (speed decreasing) effect.

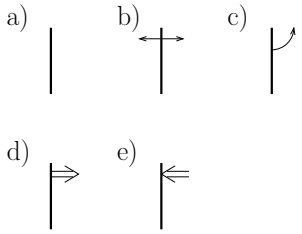


Figure 4 a) wall b) moving wall c) one-way gate d) bumper with positive effect e) bumper with negative effect.

Another component is the one-way gate. From one side, the ball passes through without any change in its motion; from the other side, it is reflected as if hitting a straight wall – behaving like a trapdoor. This is illustrated in Figure 1 b) and c). The last type of walls are parabola walls. These static walls are defined by a quadratic equation and the interval of start and end coordinates. The ball interacts with them by reflection like with a static straight wall, except the curvature has to be taken into account. We consider the tangent of the parabola at the point, where the balls hits this curve. Then, the angle of incidence of the ball is equal to its angle of reflection at this the point with respect to the tangent. Figure 4 describes the different types of walls in the problem setup.

Components

We now give a formal description of the components of our problem.

- walls : are given by the start and end positions of the wall as rational coordinates in $\mathbb{Q}^2$, see Figure 1.a).
- parabolas : are given by a quadratic function $y = a \cdot x^2 + b \cdot x + c$ where a, b, c are rational values as well as start and end values x_0 and x_1, i.e. $a, b, c, x_0, x_1 \in \mathbb{Q}$.
- one-way gates : are given by the start and end positions of the wall defined by rational coordinates. We assume that the gate opens in the clockwise direction going from the start position to the end position, see Figure 1.b) and c).
- moving walls : are given by start and end positions of the wall's initial location as rational coordinates in $\mathbb{Q}^2$. Its motion begins at a specified start time and follows a movement function over the time interval $T_0 = [0, t_1)$, with the movement function defined by a rational function in time t given by the polynomials of its numerator and denominator, i.e. the ratio of two polynomials of the time t with rational coefficients. In a second time interval $T_1 = [t_1, t_2)$, the wall moves back along a return path. We assume that after step t_2 the movement is periodically repeated. See Figure 2.
- bumpers : are given by start and end positions of the bumper given as rational coordinates, a time when the bumper effect starts, a time interval $T_0 = [0, t_1)$ with a bumper effect function for this time interval given by the quotient of two polynomials with rational coefficients depending on time t, and finally a second time interval $T_1 = [t_1, t_2)$ where the bumper rests. We assume that after step t_2 the movement is periodically repeated (Figure 3).

▶ **Definition 1** (Pinball Wizard problem). *Given a pinball with rational start and end positions, as well as its speed vector and a given set of components as described above, decide whether the ball reaches the target position.*

Given that the speed is taken as a constant, for example, the speed of light, then we obtain a similar problem to the Pinball Wizard problem called, the Ray Particle Tracing problem.

▶ **Definition 2** (Ray Particle Tracing problem). *Given a ray particle with rational start and end positions, as well as its speed vector and a given set of components as above, if we assume that walls behave the same as mirrors, decide whether the ray particle reaches the target position. The speed of the ray particle is taken as a constant, which is; the speed of light.*

The Ray Particle Tracing problem is closer to the problem analyzed by Moore in [16], as his construction for simulating a two-counter machine in 3D involves a billiard ball moving at constant speed. All proofs and observations for the Ray Particle Tracing problem can be adapted analogously to those for the Pinball Wizard problem. These adaptations are highlighted in Section 5 and discussed in detail in [2].

4 Pinball Wizard problem is Turing Complete

In this Section, we present the implementation of two independent stacks by using walls, bumper walls, moving walls, and one-way gate for the Pinball Wizard problem. For this we use the difference between the time and the position when and where the ball arrives at specific lines (i.e. a line called the starting interval or the end interval), more precisely the time and the space offset. In this Section we will present our construction using bumper walls. Note that the bumper walls can be replaced by moving walls in our constructions

(which has been discussed in [2]). Analyzing this alternative construction we can also adapt our result to the case where the ball never changes its speed, i.e. if it has reached the speed of light. This transforms the Pinball Wizard problem into the Ray Particle Tracing problem, as elaborated in [2]. Hence the Ray Particle Tracing problem is also Turing Complete if we restrict ourselves to 2D plus time.

Within our constructions we have to guarantee that the two stacks are completely independent, i.e. that push and pop operations of one stack do not change the value of the second stack (as a side effect). For the space offset stack this can be guaranteed if all the allowed paths of the ball have the same length, thus the time offset will not be affected by the stack operations of the space offset stack. Analogously for the time offset stack we have to guarantee that all allowed paths of the ball are parallel to each other. Thus, the operations of time offset stack will not influence the value of the space offset stack.

4.1 Implementing a Stack Using the Time Delay of the Pinball

The main idea to implement a stack by using a time or a space offset works as follows. Assume that the earliest time when the ball can cross a predefined (input) line is t_0 and the leftmost position on the line is s_0. And assume that the ball crosses the line at time $t_0 + \Delta t$ at position $s_0 + \Delta s$. Then we can use the offset values Δt and Δs to implement two stacks. Assume we use a second line which represents the output of a gadget implementing a stack operation. Then we use a system of walls, moving walls, bumper walls and one way gates to implement:

$$\Delta t' = \begin{cases} \frac{\Delta t}{2} & \text{to implement push 0} \\ \frac{\Delta t + 1}{2} & \text{to implement push 1} \\ 2 \cdot \Delta t & \text{to implement pop, if its output is 0} \\ 2 \cdot \Delta t - 1 & \text{to implement pop, if its output is 1} \end{cases}$$

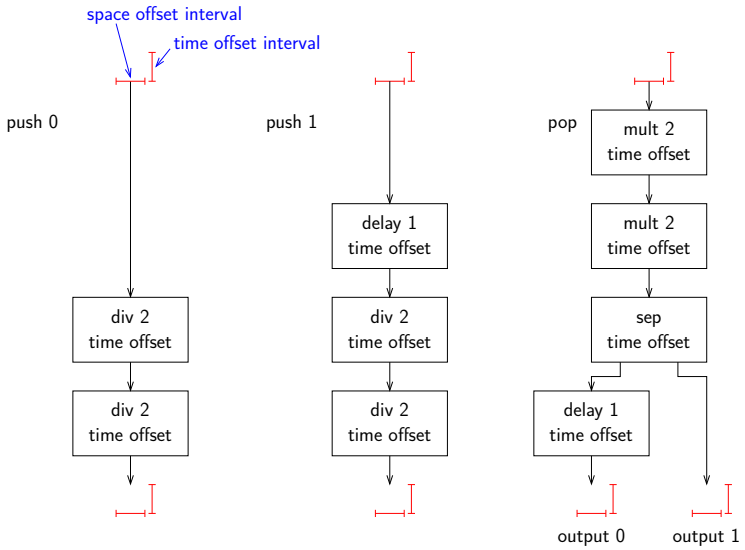


Figure 5 Illustration of the stack operations implemented by the time offset.

In the same way we can implement the stack operations using Δs for the space offset. For our time offset stack the multiplication with $\frac{1}{2}$ or with 2 is not sufficient. Since we do not assume infinite speed for the walls, a gap between the two offset intervals is therefore

necessary. This gap is implemented by multiplying the time offset two times with $\frac{1}{4}$ or 4 as required, as illustrated in Figure 5. A detailed explanation is included later in this section. We can draw a similar illustration for the space offset stack (by replacing "time" with "space", deleting the second multiplication box, and replacing the extra time offset with a space shift in Figure 5).

Let us assume that the ball enters the construction with a speed of v, then the speed will be modified by hitting the first bumper wall. By using a second bumper wall, we undo the effect of the speed modification. Hence the ball leaves the construction with a speed of v. The multiplication factor of the time offset will be generated by the modified speed within the time where the ball has a modified speed and the differed distances of the bumper walls. Recall, if the first bumper wall has a positive effect on the speed of the ball, the second has to have a negative effect and vice versa. We assume that the effect can by modified over time, analogously to acceleration. Therefore we use the acceleration of a bumper to describe this behavior of a bumper.

So the bumpers are used to manipulate the speed of the ball, and the changed speed over a given fixed distance δ_2 is used to achieve a multiplication of 2 (or $\frac{1}{2}$, resp.) with the time offset. We assume that outside of the construction blocks, the ball maintains a given speed v. This allows us to keep track of the time interval in which the ball arrives at a specific construction block.

The construction is illustrated in Figure 6. We now claim that:

▶ **Lemma 3.** *The 2D Pinball Wizard problem with one-way gates, plane and parabolic walls, moving walls, and bumpers simulates a stack using the time offset of the ball.*

Proof. We show the proof by describing how the multiplication of the time offset by 2 is implemented. Let us assume the following values where t_s is the starting time of the ball relative to a base time line. We can divide the way of the ball into three parts. Each of these parts have been described in Figures 6 (ways 1, 2 and 3).

Part 1. In the first part the ball goes from the starting line to the first moving wall.

- δ_1 is the distance between the starting line of the ball and the start line of the moving interval of the first moving wall.
- v is the initial speed of ball and its final speed.
- The ball reaches first the bumper wall at time step $t_1(t_s)$. Note that $t_1(t_s) = t_s + \frac{\delta_1}{v}$. We use $t_1 = t_1(0)$.
- $a_1(t)$ is the acceleration of the first bumper wall. We assume that the acceleration is 0 at time $t \leq t_1$. The effect of the bumper wall will be $v_1(t) = (t - t_1) \cdot a_1(t)$.
- $v_{b,1}(t_s)$ is the speed of the ball after the collision with the first bumper wall. Note that $v_{b,1}(t_s) = v - v_1(t_1 + t_s)$.

Note that

$$t_1 = \frac{\delta_1}{v} \qquad \text{and} \qquad v_1(0) = v \ .$$

Part 2. In the second part the ball goes from the the first bumper wall to the second bumper wall.

- δ_2 is the distance between the first bumper wall and the second bumper wall.
- $v_{b,1}(t_s)$ is the speed of ball within this part.
- The ball reaches the second bumper wall at time step $t_2(t_s)$. Note that $t_2(t_s) = t_1(t_s) + \frac{\delta_2}{v_{b,1}(t_s)} = t_s + \frac{\delta_1}{v} + \frac{\delta_2}{v_{b,1}(t_s)}$. We use $t_2 = t_2(0)$.

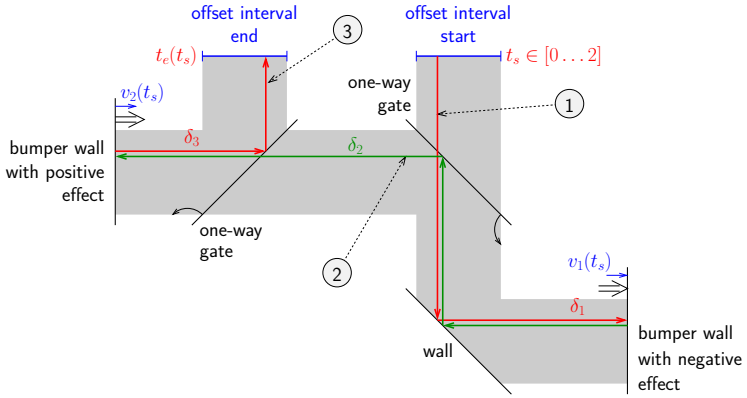


Figure 6 Multiplying the time offset by a factor using bumpers: The first red way (way 1) from the input (at offset interval start) to the right hand side bumper wall illustrates the the length δ_1. The green way (way 2) between the two bumper walls illustrates the the length δ_2. This is the only way where the ball has a changed speed. The second red way (way 3) from the left hand side bumper wall to the output (at offset interval end) illustrates the the length δ_3. The way between the two bumper walls is used to implement the multiplication of the time offset by a constant. The vertical shift of the 3 ways is used to increase readability.

- $a_2(t)$ is the acceleration of the second bumper wall. We assume that the acceleration is 0 at time $t \leq t_2$. The effect of the bumper wall will be $v_2(t) = (t - t_2) \cdot a_2(t)$.
- $v_{b,2}(t_s) = v$ is the speed of the ball after the collision with the second bumper wall. Note that $v_{b,2}(t_s) = v_{b,1}(t_s) + v_2(t_2(t_s))$.

Note that

$$t_2 = \frac{\delta_1 + \delta_2}{v} \qquad \text{and} \qquad v_{b,2}(0) = v_{b,1}(0) \;=\; v \;.$$

Part 3. In the third part the ball goes from the the second bumper wall to the end line.

- δ_3 is the distance between the second bumper wall and the end line of the ball.
- The ball reaches the end line at time step $t_c(t_s)$. Note that $t_e(t_s) = t_2(t_s) + \frac{\delta_3}{v}$. We use $t_e = t_e(0)$.

Note that

$$t_e = \frac{\delta_1 + \delta_2 + \delta_3}{v} \qquad \text{and} \qquad t_e(t_s) = t_s + \frac{\delta_1 + \delta_3}{v} + \frac{\delta_2}{v_{b,1}(t_s)} \;.$$

Our goal is now to determine functions for the acceleration $a_1(t_s)$ and $a_2(t_s)$ such that $t_e(t_s) = 2 \cdot t_s + c$ for a value $c = t_e$ which does not depend in t_s. This leads to

$$t_s + \frac{\delta_2}{v} \;=\; \frac{\delta_2}{v_{b,1}(t_s)}$$

$$\text{and therefore} \qquad v_{b,1}(t_s) \;=\; \frac{v \cdot \delta_2}{v \cdot t_s + \delta_2} \;.$$

Hence

$$a_1(t_1 + t_s) \;=\; \frac{v - v_{b,1}(t_s)}{t_s} \;=\; \frac{v^2}{v \cdot t_s + \delta_2}$$

$$a_2(t_2 + 2 \cdot t_s) \;=\; \frac{v - v_{b,1}(t_s)}{2 \cdot t_s} \;=\; \frac{v^2}{2 \cdot (v \cdot t_s + \delta_2)}$$

◀

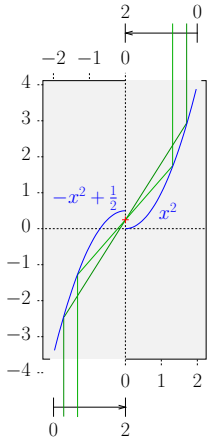


Figure 7 After performing a multiplication step, the order of the offset is reversed (the zero offset is moved from the left of the pinball interval to the right of the interval). This tool moves the pinball to its original order, i.e. the zero offset is moved back to the left side of the interval. Depending on the widths of the pinball interval, one might have to implement a divergent version of this tool, i.e. with x coordinates going from -4 to 4 instead of the presented version with x coordinates going from -2 to 2. Within the following construction only offsets between 0 and 2 (excluding 0 and 2) are used.

For the multiplication of the time offset by $\frac{1}{2}$ we exchange the two bumpers and the analysis will follow the same ideas as above.

Note that at the output interval the order of the space offset is reversed. To correct the order we have to add the construction of Figure 7 at the output interval of the time offset multiplier.

It is also possible to implement a multiplier for the time offset by using only moving walls (see Figure 8). The analysis can be found in [2]. The implementation of the multiplier by moving walls can be adapted such that it can be used to handle the case where the ball does not change its speed (leading to the Ray Particle Tracing Problem). If we simulate bumpers with moving walls the movement of both walls has to be coordinated in such a way that the different speeds and delays add up to the same effect as two bumpers.

Finally we have to describe a gadget for adding a constant delay to the ball and a gadget for separating early balls (arriving at a specific position with a time offset of $[0 \dots 1)$) from late balls (arriving at a specific position with a time offset of $[1 \dots 2)$). The constant delay can easily be generated by using some 45° walls. Hence we will now focus on the construction of the time offset separator.

The main idea of the time offset separator is to use a moving wall which leads a ball arriving with a time offset $[0 \dots 1)$ in one distinct direction and a ball arriving with a time offset $[1 \dots 2)$ in a second distinct direction. Since we do not assume that a wall can move with an infinite speed, we have to implement a gap between the two offset intervals. This can be implemented be doubling the multipliers, i.e. instead of multiplying the time offset once with $\frac{1}{2}$ (or with 2, respectively), we multiply the time offset two times with $\frac{1}{2}$ (or with 2, respectively). Hence we multiply the time offset either with $\frac{1}{4}$ or with 4.

As illustrated in Figure 5 the time offset separator only occurs after the multiplication with 4 within the pop operation. Thus we can assume that the ball will arrive with a time offset within the interval $[0 \dots 2)$ (instead of within the standard interval $[0 \dots 1)$). Using double multiplication guarantees that the ball will either arrive with a time offset in $[0 \dots \frac{1}{2})$ or with a time offset in $[1 \dots \frac{3}{2})$.

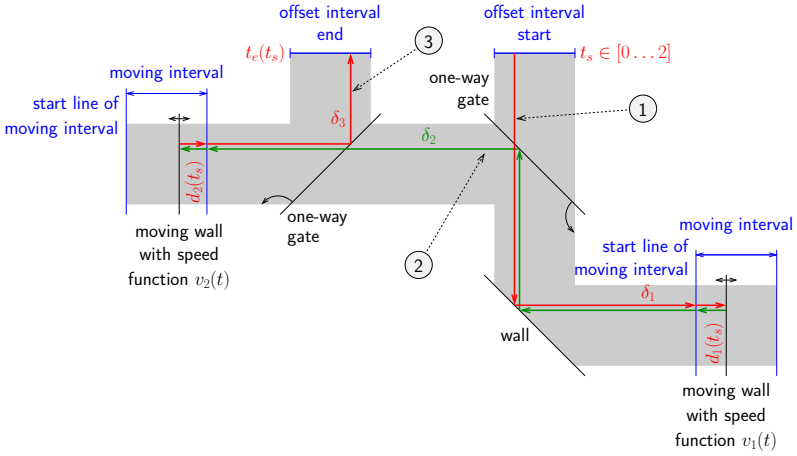


Figure 8 As is the case with the multiplication using bumpers, we can split the way from the starting interval to the end interval into 3 part: The first red way (way 1) from the input (at offset interval start) to the right hand side moving wall, the green way (way 2) between the two moving walls, and the second red way (way 3) from the left hand side moving wall to the output (at offset interval end). Way 2 is used to implement the multiplication of the time offset by a constant.

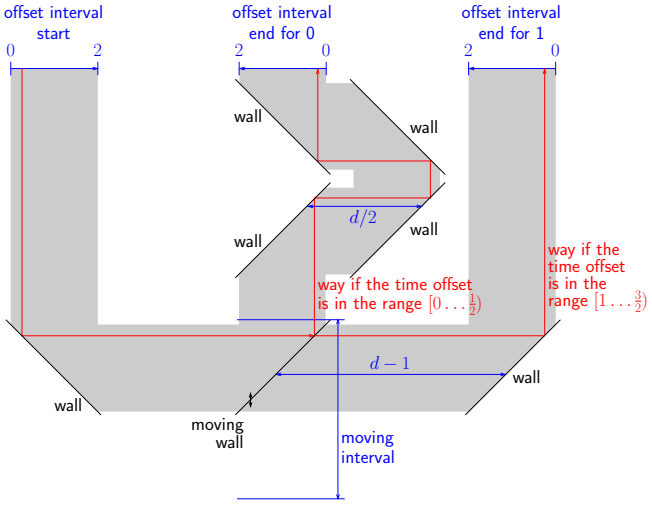


Figure 9 Construction for separating the ball arriving with a time offset in the time interval $[0 \dots \frac{1}{2})$ from the ball arriving with a time offset in the time interval $[1 \dots \frac{3}{2})$. Using the distance of $d-1$ instead of d shifts the base of the time interval of the second interval $[1 \dots \frac{3}{2})$ from 1 to 0. The time gaps in the offset between $\frac{1}{2}$ to 1 (and from $\frac{3}{2}$ to 2) are necessary to give the moving walls the time either to move out of the way of the ball or to move into its initial position.

This allows the separating moving wall to move out of the way and to move back. The construction is illustrated in Figure 9. We refer the reader to [2] for a more detailed analysis.

4.2 Implementing a Stack Using the Space Offset of the Pinball

The space offset stack can be implemented using a system of plane walls, one-way gates and parabola walls. Note that the distance the ball has to pass to reach the focus is independent of its starting position on the starting interval. This distance is given by the distance from the starting line segment to the point where the ball hits the parabola wall plus the distance between this point to the focus.

Let us assume that a ball going down in the direction of the y-axis starts at an arbitrary point ℓ on the line segment defined from point $(0, h)$ to point $(2, h)$ at time t. Assume again that the ball bounces off a parabola $(x, a \cdot x^2)$ with focus $(0, \frac{1}{4a})$. If the ball starts at point (ℓ, h) then the distance traveled by the ball to the focus is given by

$$h - a \cdot \ell^2 \quad + \quad a \cdot \ell^2 + \frac{1}{4a} \quad = \quad h + \frac{1}{4a}$$

which is independent from the starting point at the starting line segment (see Figure 10). If we assume that within each time unit the ball can pass through one space unit, then within a construction like in Figure 11, it will reach the focus at time $t + h + \frac{1}{4a}$, independent from ℓ.

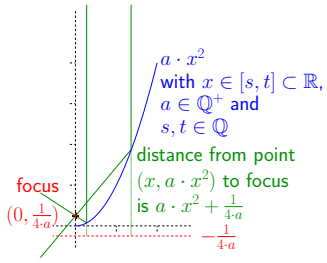


Figure 10 A parabola with points $(x, a \cdot x^2)$ and focus $(0, \frac{1}{4a})$.

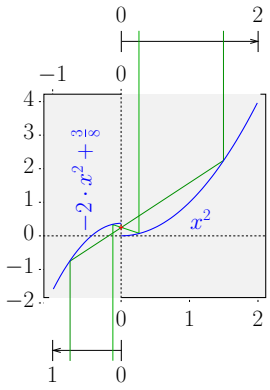


Figure 11 Multiplication of the offset by $\frac{1}{2}$ is a main tool for implementing a push operation. Here we show the reverse ordering when two separate balls enter the starting interval line at different positions.

▶ **Lemma 4.** *The Pinball Wizard problem can simulate a stack which is implemented by the space offset of the ball within a system of parabola walls.*

Proof. We achieve push and pop operations using the multiplication of the offset by 2^{-1} or by 2. We assume starting interval between 0 and 2. For the multiplication of the offset by $\frac{1}{2}$, two parabolas are placed such that the second parabola is flipped as seen in Figure 11. Analogously, multiplying the offset by 2 can be done using a similar construction (see [2]). As with the time offset stack, both constructions reverses the ordering of space offset of the ball. Thus a similar construction as shown in Figure 7 can be used to correct the ordering.

Larger offsets are allowed to deal with pushing 1 to or popping 1 from the stack, as both operations require the addition of a 1 to the offset (that is, a shift in the offset interval).

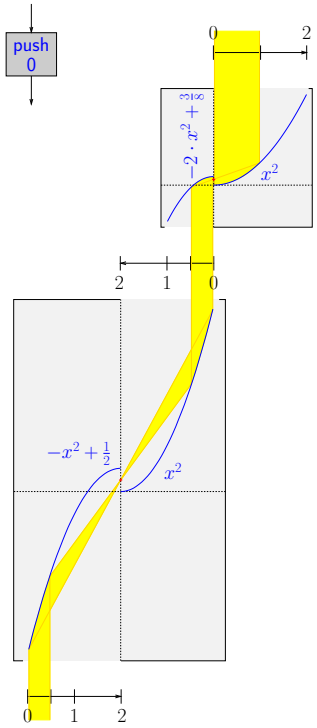


Figure 12 Multiplying the offset from the interval $[0, 1)$ by $\frac{1}{2}$ implements a push of the Boolean value 0 to the stack. The yellow area represents area of possible offsets.

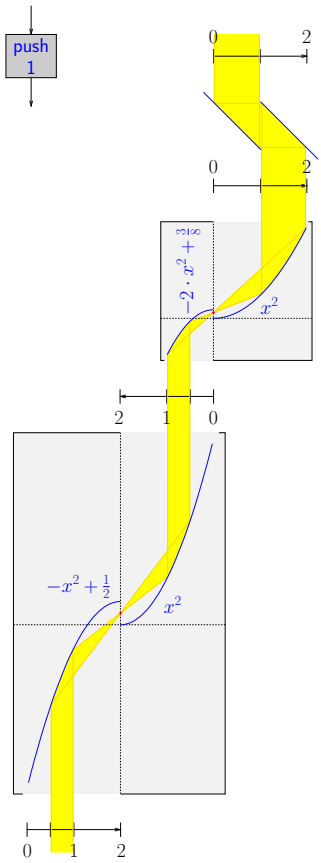


Figure 13 Shifting the offset from the interval $[0, 1)$ to the interval $[1, 2)$ and multiplying the resulting offset by $\frac{1}{2}$ implements a push of the Boolean value 1 to the stack. The yellow area represents area of possible offsets.

For push 0 we multiply the space offset by $\frac{1}{2}$ (see Figure 12). To push 1 we first shift the starting offset interval from $[0 \ldots 1)$ to $[1 \ldots 2)$ before multiplying the space offset by $\frac{1}{2}$ (see Figure 13). For pop 1, after multiplying by 2, the possible output offset values are split in two intervals, each going from 0 to 1 as seen in Figure 14. Thus if the top entry of the stack is a 0, the ball with the modified offset will show up on the left exit. In the case where it is a 1, the ball will accrue to the right exit. To separate these two cases we have to split the interval after the multiplication into two intervals to get the resulting stack and the output of the pop operation.

By assuming the bottom of the stack always contains at least two 1 digits, the offset can never have the values 0, $\frac{1}{2}$, and 1. To guarantee that these edge cases never occur we can assume a bottom symbol on the stacks which will never be removed. ◀

The full proof as well as constructions associated with Lemma 4 are given in [2].

Note that both pop operations (for the time offset stack as well as for the space offset stack) results in two potential output intervals (one for the pop result 0 and one for the pop result 1). When we would like to continue with our simulation of the two-stack PDA we have to combine the two intervals again. For this we can use a one-way gate, which guarantees a

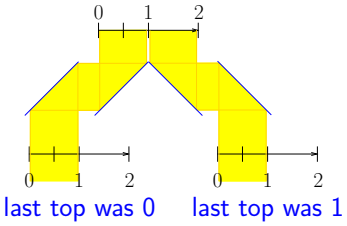


Figure 14 Construction for separating the arrival of the ball within the space interval $(0 \ldots 1)$ from the ball's arrival within the space interval $(1 \ldots 2)$.

perfect overlapping of the two intervals (the 0 offset of the first interval will overlap with the 0 offset of the second interval and the 1 offset of the first interval will overlap with the 1 offset of the second interval).

To illustrate the behavior of space as well as time offset intervals w.r.t. push and pop operations we will showcase a short example in the following :

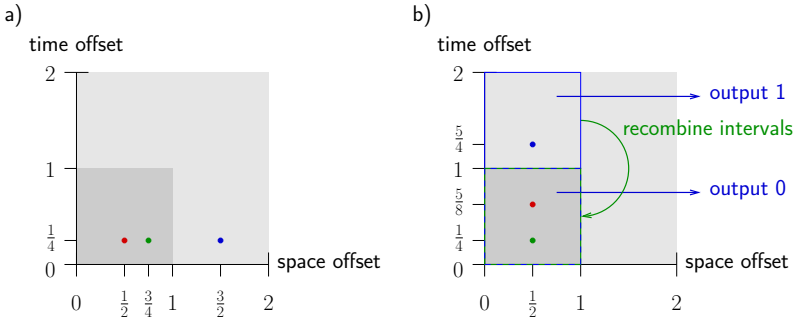


Figure 15 Behaviour of the time and space offset intervals according to the (a) push(1) and (b) pop operations. The red dot denotes the initial position of the stack, the blue dot shows the intermediate position and the green dot shows the final position of the stack.

For the behaviour of the offset intervals according to stack operations (see Figure 15 a), starting with the stack values 01 (represented by the binary value 0.01_2) for the time offset stack and 1 (represented by the binary value 0.1_2) for the space offset stack (red dot), we begin by performing a push(1) operation to the space offset stack: First add 1 to the value of the space offset stack (blue dot), and then multiply by 2^{-1} (green dot). This leads to the stack content 11 (represented by the binary value 0.11_2).

In Figure 15 b), we start with stack values 101 (represented by the binary value 0.101_2) for the time offset stack and 1 (represented by the binary value 0.1_2) for the space offset stack (red dot) and perform a pop operation on the time offset stack as follows: First multiply the value of the time offset stack by 2 (blue dot). If the result is in the range between 1 and 2, the output will be 1, otherwise if the result is in the range between 0 and 1, the output will be 0. Finally we recombine the two possible intervals (by moving the upper interval to map on the lower interval, resulting in the green dot). This leads to the stack content 01 (represented by the binary value 0.01_2) in the time offset stack. Note that all the operations should not modify the space offset value.

One should highlight here that from Figure 10, it can be seen that for every offset of the ball that might occur within each of the push 0, push 1 and pop constructions, the ball will always need the same amount time to go from the horizontal start line to the finish line.

This is possible because an extra delay can be introduced on the ball's movement by an additional substructure such that its time of arrival at the end line is well-defined. Hence, the operations of the space offset stack is independent of the time offset stack.

▶ **Lemma 5.** *The 2D Pinball Wizard problem with one-way gates, walls and moving walls simulates a pushdown automaton where the stack is represented by a time offset such that each step is represented by a constant number of reflections.*

Recall that a pop operation leads to 2 possible alternative ways the ball might go (see Figure 9 and 14). The two alternative ways represent the two possible different calculation steps of the simulated two-stack pushdown automaton depending on the top element of one stack. To continue with the simulation the two alternative ways have to rejoined. This will be done by the construction presented in Figure 16.

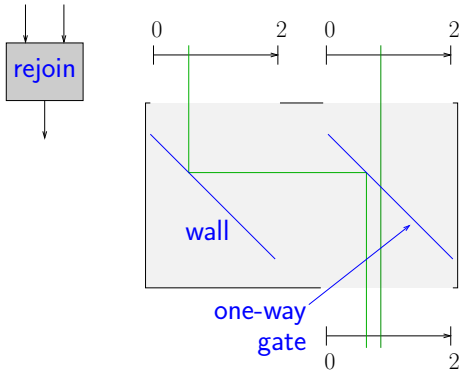


Figure 16 A simple rejoin tool to be used after the splitting at a pop operation.

Now, using the proofs of Lemma 3, Lemma 4 as well as Lemma 5 (proof available in [2]), we conclude that

▶ **Theorem 6.** *The 2D Pinball Wizard problem with one-way gates, plane, parabolic walls and moving walls simulate a two-stack pushdown automaton (PDA) such that each step is represented by a constant number of reflections. This also holds if we replace the moving walls by bumpers.*

Since a two-stack PDA can simulate any deterministic Turing machine, we conclude the following:

▶ **Corollary 7.** *Deciding the 2D Pinball Wizard problem with one-way gates, plane, parabolic and moving walls is as hard as solving the Halting problem.*

5 Ray Particle Tracing Problem is Turing Complete

Let us now assume that bumpers and moving walls do not change the speed of the ball, i.e. our ball has always the same speed v. This is for example the case if we assume that the ball has the speed of light, leading to the Ray Particle Tracing problem. Note that bumpers behave like usual walls if the speed of a ball cannot be changed. The only effect of a moving wall on the ball might be a change in the distance which the ball has to go. Within this problem we usually use mirrors instead of walls. However, for uniformity in the analyses, we stay with the word "wall".

Since the distance a ball has to travel influences the traveling time one could implement multipliers for the space and time offset only by using simple (non-moving) and by moving walls. For the implementation of the space offset we do not need any changes in our construction. For dealing with the time offset stack some changes have to be done to original construction. If the moving wall can change the speed of the ball, we have to analyze the changed speed and the changed travel distance (see the construction for such a multiplier in Figure 8 and its analysis in [2]).

In the special case of a ball traveling with the speed of light, one could only assume that such a multiplier multiplies the time offset with a value of $1 + \varepsilon$ or $1 - \varepsilon$, respectively, for a small but constant value of ε. Otherwise the walls have to move with nearly the half of the speed of light. To reach a multiplication with 2 or $\frac{1}{2}$ one has to iterate the multiplier (for the analysis see [2]). On the other hand, if the ball never changes its speed by a collision with a moving wall, we do not have to undo such changes. Thus the resulting construction for the multiplication of the time offset looks simpler than the construction presented in the previous section.

6 Conclusions and Open Problems

Inspired by pinball machines, the Pinball Wizard problem involves navigating a maze composed of stationary and moving walls, one-way gates, parabola walls and bumper walls. The objective is to determine whether a pinball can reach a designated target. We show that the Pinball Wizard problem in two dimensions can simulate a two-stack PDA, which is equivalent in power to a single-tape deterministic Turing machine. Thus, solving instances of this problem is at least as hard as deciding the Halting problem.

While the billiard problem with static walls – also known as ray tracing with mirrors – is known to be Turing-complete in three dimensions, as established independently by Moore [16] and Reif et al. [20], our work may contribute to resolving the open question of whether Turing-completeness can also be achieved in two dimensions.

Another interesting direction for future research is to investigate whether a simplified model with a reduced set of components can still achieve Turing-completeness. Currently, our construction relies on idealized assumptions such as arbitrary precision and perfect reflection. It remains an open question how the computational complexity would change under more realistic physical constraints, including damping, friction, finite ball diameters and random disturbances. Future work could also explore whether a smaller set of components suffices for the Pinball Wizard problem specifically and what precision bounds are necessary for reliable computation.

It is an open problem whether the presented constructions can be modified such that no one-way gate for rejoining the different simulation steps after the splitting at a pop operation are necessary anymore. This leads to the question of simulating reversible Turing Machines by the presented Pinball Wizard problem.

References

1 Andrew Adamatzky and Jérôme Durand-Lose. Collision-based Computing. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing, Section VII: Broader Perspective — Alternative Models of Computation*, pages 1949–1978. Springer, 2012. doi:10.1007/978-3-540-92910-9_58.

2 Rosemary Adejoh, Andreas Jakoby, Sneha Mohanty, and Christian Schindelhauer. How pinball wizards simulate a turing machine, 2025. arXiv:2510.02560.

3 Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press, 2004. doi:10.1201/9780429487309.

4 Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. *Journal of Theoretical Biology*, 309:121–133, 2012. doi:10.1016/j.jtbi.2012.06.017.

5 Tanay Chattopadhyay and Dilip Kumar Gayen. Optical xor-xnor logic circuits using mechanical movable mirrors. In *2021 Devices for Integrated Circuit (DevIC)*, pages 65–70, 2021. doi:10.1109/DevIC50843.2021.9455909.

6 Stephen Cook and Ravi Sethi. Storage requirements for deterministic / polynomial time recognizable languages. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 33–39, New York, NY, USA, 1974. Association for Computing Machinery. doi:10.1145/800119.803882.

7 Erik Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. In *Algorithms and Complexity*, pages 177–195, April 2017. doi:10.1007/978-3-319-57586-5_16.

8 Jérôme Durand-Lose and Andrew Adamatzky. *Computing Inside the Billiard Ball Model*, pages 135–160. Springer London, London, 2002. doi:10.1007/978-1-4471-0129-1_6.

9 Michal Forišek. Computational complexity of two-dimensional platform games. In Paolo Boldi and Luisa Gargano, editors, *Fun with Algorithms*, pages 214–227, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13122-6_22.

10 Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of theoretical physics*, 21(3-4):219–253, 1982. doi:10.1007/BF01857727.

11 Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970. URL: http://www.jstor.org/stable/24927642.

12 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Co., New York, 1980. doi:10.1016/0096-0551(80)90011-9.

13 Manuel Lafond. The complexity of speedrunning video games. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:19, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FUN.2018.27.

14 Norman Margolus. Physics-like models of computation. *Physica D: Nonlinear Phenomena*, 10(1):81–95, 1984. doi:10.1016/0167-2789(84)90252-5.

15 Sneha Mohanty and Christian Schindelhauer. Cryptography based on 2d ray tracing. In *2025 9th International Conference on Cryptography, Security and Privacy (CSP)*, pages 33–40, 2025. doi:10.1109/CSP66295.2025.00014.

16 Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64:2354–2357, May 1990. doi:10.1103/PhysRevLett.64.2354.

17 Toshiyuki Nakagaki. Smart behavior of true slime mold in a labyrinth. *Research in Microbiology*, 152(9):767–770, 2001. doi:10.1016/S0923-2508(01)01259-1.

18 Marian Boylan Pour-el and Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17(1):61–90, 1979. doi:10.1016/0003-4843(79)90021-4.

19 Stephen R Pring and Christopher J Budd. The dynamics of a simplified pinball machine. *IMA Journal of Applied Mathematics*, 76(1):67–84, 2011. doi:10.1093/imamat/hxq064.

20 John H. Reif, J. Doug Tygar, and A. Yoshida. Computability and complexity of ray tracing. *Discrete & Computational Geometry*, 11:265–288, 1994. doi:10.1007/BF02574009.

21 Ryuhei Uehara. Computational complexity of puzzles and related topics. *Interdisciplinary Information Sciences*, 29(2):119–140, 2023. doi:10.4036/iis.2022.R.06.

22 Liang Zhang. Computing naturally in the billiard ball model. In *International Conference on Unconventional Computation*, pages 277–286. Springer, 2009. doi:10.1007/978-3-642-03745-0_29.

23 Martin Ziegler. Physically-relativized church–turing hypotheses: Physical foundations of computing and complexity theory of computational physics. *Applied Mathematics and Computation*, 215(4):1431–1447, 2009. Physics and Computation. doi:10.1016/j.amc.2009.04.062.

24 Tadeu Zubaran and Marcus Ritt. Agent motion planning with pull and push moves. In *Anais do VIII Encontro Nacional de Inteligência Artificial*, pages 358–369, Porto Alegre, RS, Brasil, 2011. SBC. URL: https://sol.sbc.org.br/index.php/eniac/article/view/33707.

A Simulating a DTM

Here, we will sketch how the Pinball Wizard problem and the Ray Particle Tracing problem can be used to simulate the computation of a 1-tape DTM (equivalently, a two-stack PDA). Hence both problems are Turing complete. Hence both problems are at least as hard as the Halting problem.

Assume a configuration of a 1-tape DTM (equivalently, a two-stack PDA). To represent the tape content left of the head we can use the time offset stack and to represent the tape content right of the head we can use the space offset stack. Given a 1-tape DTM with a binary input string w, let $S^1 = \langle q_0, s_0^1, w\rangle$ and $S^2 = \langle s_0^2\rangle$ be the initial configuration of the space and time offset stacks, respectively where q_0 is the initial state of the DTM. We assume that the states of the DTM are represented by binary numbers of length ℓ. s_0^1 and s_0^2 are the bottom symbols encoded in the stacks.

At the start of every step in the simulation, if the current state is q_i and a_i is the symbol in the cell read by the DTM. Then we assume that a_i, all the elements to the right of a_i and q_i are in the space offset stack and the elements to the left of a_i are in the time offset stack.

In each step, the current state q_i and cell symbol a_i are read from the space offset stack via a sequence of pop operations, each of which will result in a component which uniquely describe the starting point of the next step. Recall that each pop operation yields a binary decision tree over the possible output intervals regions, see Figure 14. In order to perform the transition function, we use a binary decision tree, where we branch out using the pop operation for every of the possibilities defined by the starting state and symbol read from the space offset stack. Since this is a DTM, the next state q_j, the new cell symbol a_j, and the direction τ in which the head will move are uniquely defined. Next, we write a_j to the current cell and move the tape head again by pop operations and push operations. Finally, we write the encoding of the new state q_j to the space offset stack, combine the space offset intervals and time offset intervals (recall that the separators after the multiplications with 2 result in several alternative offset intervals) to a single unique output interval. Hence, the branches of the decision tree are combined again into a single space and time offset using one-way gates. The tool for combining intervals consisting of a wall and a one-way gate is shown in Figure 16.

If necessary, additional delay on the particle can be introduced by strategically placed walls. Now we can start with the next simulation step.

For the moving walls we schedule the return time to the initial position according to the constant simulation time of each step of the Turing machine. So, we can reuse them for simulating in the next step.

Distributed Games with a Central Decision Maker

Bharat Adsul
Indian Institute of Technology Bombay, India

Nehul Jain
Indian Institute of Technology Bombay, India

Abstract

We study distributed games played on non-deterministic asynchronous automata which feature a central decision maker process that participates in all key decision making tasks. In these partial-information games, processes use their causal past to respond to scheduling choices made by the scheduler and cooperatively strategize as a team to achieve the winning objective. We show that the problem of deciding the existence of a distributed winning strategy is efficiently solvable for global safety and local parity objectives. We provide algorithmic solutions that match their computational hardness. We formulate the notion of a finite-state distributed strategy which allows to quantify its distributed memory requirements. For the aforementioned objectives, we establish that finite-state distributed winning strategies always exist. In fact, we provide novel constructions of such winning strategies which are shown to have almost optimal amount of distributed memory. We also show that a natural extension of the model with two decision making processes is undecidable.

2012 ACM Subject Classification Theory of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases Mazurkiewicz traces, models of concurrency, distributed synthesis, game-theoretic models, asynchronous automata, distributed decision-making

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.5

1 Introducion

The design and analysis of concurrent programs and distributed protocols have always presented significant challenges. With the increasing pervasiveness of distributed applications, it is natural to investigate the automated synthesis of such protocols from their specifications. The foundational work on distributed synthesis [19] demonstrated the general undecidability of the problem when processes have access to only purely local information and limited communication capabilities. However, recent works [9, 14, 10, 18, 8, 11, 1] have introduced richer models where processes have access to their entire *causal past*, broadening the scope of problems that can be tackled. In particular, [9] introduced the notion of causal past and addressed the distributed controller synthesis problem in series-parallel systems and established that controlled reachability is decidable. The work [14] introduced systems with connectedly communicating processes and proved that the MSO theory in this setting is decidable which implies that the associated distributed synthesis problems are decidable.

Two more recent and relevant lines of work are Petri games and asynchronous games. Petri games [8] are distributed games played on Petri nets, where tokens represent players. The places in the underlying Petri net are divided into system and environment places, and tokens in system places make decisions based on their causal history. Several interesting classes of Petri games have been shown to be decidable [8, 7] but it was recently shown in [6] that Petri games with global winning conditions are undecidable, even with only two system players and one environment player.

Asynchronous games [10] are played on deterministic asynchronous automata and allow processes to control actions based on their causal past. These games are shown to be decidable in the interesting setting of acyclic architectures [10, 18] whose underlying process-

© Bharat Adsul and Nehul Jain;
licensed under Creative Commons License CC-BY 4.0

communication graph is acyclic. More general formulations like decomposable games [11] have also been identified and proven to be decidable. However, the undecidability results for asynchronous games with simple objectives such as local reachability/termination/deadlock-freeness, even in systems with six processes, from [12] highlights the complexity of distributed synthesis problems.

Clearly it is desirable to identify different settings of the distributed synthesis problems which are interesting, practically well motivated and decidable. In this work, we consider the distributed setting in which there is a designated process called the *central decision maker*. We refer to such distributed systems as CDM systems in which all the key decisions are taken in the presence of this process. In many server-client architectures the server process is primarily responsible for maintaining overall integrity and ensuring that the demands of the client processes are met. These systems serve as prototypical examples of CDM systems. A distributed version control system such as SVN may also be viewed as a CDM system in which a single authoritative main copy is maintained at the central repository. Users make concurrent changes to local copies; in order to effect these changes to the main copy, they acquire an exclusive access to the central repository using a "lock". One can model this locking mechanism by a central decision maker process which synchronizes with different user processes. Another illustrative example of a CDM system is the working of an institution/organization consisting of multiple agents with a designated *head*. Various committees are formed to address different aspects of the organization, expedite work and increase overall efficiency. An agent is typically a member of several such committees. These committees conduct procedural meetings from time to time to discuss, gather relevant information and propose solutions. However, all the key decisions are taken in only those committee meetings in which the head participates. The head's presence in all decision making activities ensures consistency and alignment with the organizational goals. Note that concurrent meetings of different committees are allowed in this example.

In this paper, we formulate and study a distributed CDM game which is played on a *non-deterministic* asynchronous automaton A involving a team $\mathscr{P}$ of processes and assume the presence of a central decision maker, a special process in $\mathscr{P}$ which participates in *every* action which is *not* deterministic. In these games, the environment manifests itself as a *scheduler* of actions. Each scheduling decision gives rise to an *event* which is distributed across the participating processes. These scheduling *events* inherit a natural partial order in which, for every process, the events in which this process participates, are totally ordered. The processes participating in a scheduled event must respond to the corresponding scheduling choice, purely based on their *collective causal past*, by providing a *matching local transition* of A. It is important to keep in mind that the environment is simply an *abstract* scheduler of (possibly concurrent) actions/events and it *does not reveal* any information about concurrent scheduling choices. In particular, the causal past of the central decision maker contains information only about some scheduling events and gets updated with possibly *unbounded* information about concurrent scheduling events as the play proceeds. Further, the causal past of any other process may *not* contain complete information about the decisions taken by the central decision maker. It has access to only those decisions which it learns directly or indirectly through other processes and is oblivious to concurrent scheduling decisions made by the central decision maker.

We show that the key problem of solving a CDM game, that is, deciding the existence of a distributed winning strategy, is efficiently solvable for global safety and local parity winning objectives. We also provide hardness results for these problems which demonstrate that our algorithmic solutions are optimal. More precisely, we show that the problems of solving

CDM games for the above objectives are EXPTIME-complete. Our EXPTIME-completeness result for global safety CDM games has some resemblance with a similar result for the Petri game model with one system token studied in [7]. It is important to note that [7] is only concerned with safety objectives. In contrast, we provide a uniform method to address both safety as well as parity objectives. Towards this, we build upon the theory of Mazurkiewicz traces and the theory of two-player games. More precisely, given a CDM game, we associate with it a natural two-player full-information game. Our key result allows to "extract" from a winning strategy in this two-player game, a distributed winning strategy in the CDM game. This is achieved by crucially relying on novel *special* linearizations of traces that we develop in this work. We believe that these special linearizations have much wider applicability.

This work also formulates a natural notion of a *finite-state distributed memory strategy*. Such a strategy is modelled as a deterministic asynchronous automaton in which each process keeps track of some additional key information in its *local memory states* and the responses to scheduling choices are computed with the help of the joint-memory states. The amount of additional local memory provides a measure of the *distributed memory complexity* of such a finite-state strategy. Recall that, in the standard full-information two-player graph games central to reactive synthesis (see [13]), safety and parity objectives can be achieved by positional/zero-memory winning strategies. In contrast, non-trivial *distributed* memory is required, in general, for winning global safety and local parity objectives in the CDM setting. For the safety and local parity objectives on CDM games, we establish that existence of distributed winning strategies implies existence of finite-state distributed winning strategies. In fact, we provide a novel construction of a finite-state winning strategy in which each process keeps in its local memory the latest global-state that is part of its causal past. We use the gossip automaton from [17] to maintain and update this information. It turns out that the distributed memory complexity of our construction is exponential in $|\mathscr{P}|$ - the number of processes. We also show that this exponential dependence on $|\mathscr{P}|$ is unavoidable.

Let us mention that our analysis applies to the more general setting of distributed games in which the "decision" events are causally ordered. In this general setting, any pair of actions, which are *not deterministic*, must have *some* process participating in both of them. We thus rule out concurrent/independent actions which are not deterministic. Clearly, this setting includes CDM games as the central decision maker participates in *all* actions which are not deterministic. Throughout the paper, we mainly address the CDM setting for convenience. Our final result concerns an extension of the CDM model which permits two decision making processes. In this natural extension, in every action which is not deterministic, at least one of the two decision makers participates. We build on the technical machinery of [12] to show the surprizing result that safety games with two decision makers are undecidable.

2 Preliminaries

In this section, we set up basic notations about (Mazurkiewicz) traces (see [4, 3, 16]). A trace is a well-established model of a concurrent behaviour in which the causality and concurrency information between events is represented in the form of a labelled partial order.

Let $\mathscr{P}$ be a finite non-empty set of processes, and let i, j range over $\mathscr{P}$. We write $\{X_i\}$ for a $\mathscr{P}$-indexed family $\{X_i\}_{i\in\mathscr{P}}$. A *distributed alphabet* over $\mathscr{P}$ is a family $\widetilde{\Sigma} = \{\Sigma_i\}$ of finite sets. Let $\Sigma = \bigcup_{i\in\mathscr{P}} \Sigma_i$ be the total alphabet, i.e., the set of all actions. For $a \in \Sigma$, we set $\mathrm{loc}(a) = \{i \in \mathscr{P} \mid a \in \Sigma_i\}$ to be the set of processes which participate in it. We let $I = \{(a, b) \in \Sigma \times \Sigma \mid \mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset\}$ and $D = (\Sigma \times \Sigma)\backslash I$ be the induced *independence* and *dependence* binary relations on Σ respectively. For a poset $(X, \leq)$ and $Y \subseteq X$, we define $\downarrow Y = \{x \in X \mid x \leq y \text{ for some } y \in Y\}$; for $x \in X$, we set $\downarrow x = \downarrow\{x\}$ and $\Downarrow x = \downarrow x \setminus \{x\}$.

▶ **Definition 1.** *A trace over $\widetilde{\Sigma}$ is a Σ-labelled poset $t = (E, \leq, \lambda)$ where,*

- *E is a (possibly infinite) set of* events *and $\lambda : E \to \Sigma$ is a labelling function.*
- *$\leq$ is a partial order on E such that*
  1. *for each $e, e' \in E$, $(\lambda(e), \lambda(e')) \in D$ implies $e \leq e'$ or $e' \leq e$.*
  2. *for each $e, e' \in E$, $e \lessdot e'$ implies $(\lambda(e), \lambda(e')) \in D$, where $e \lessdot e'$ if $e < e'$ and for each e'' with $e \leq e'' \leq e'$, either $e'' = e$ or $e'' = e'$.*
  3. *for each $e \in E, \downarrow e$ is finite.*

Let $t = (E, \leq, \lambda)$ be a trace over $\widetilde{\Sigma}$. The elements of E are referred to as *events* in t. For $e \in E$, $\mathrm{loc}(e)$ abbreviates $\mathrm{loc}(\lambda(e))$. For $i \in \mathscr{P}$, the set of *i-events* in t is $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$, i.e., the events in which process i participates. Note that the first condition in the definition of a trace implies that E_i is totally ordered by $\leq$. We write $\mathrm{loc}(t) = \{i \in \mathscr{P} \mid E_i \neq \emptyset\}$ to denote the set of processes which participate in *some* event in t. Note that $\mathrm{loc}(t) = \cup_{e \in E} \mathrm{loc}(e)$.

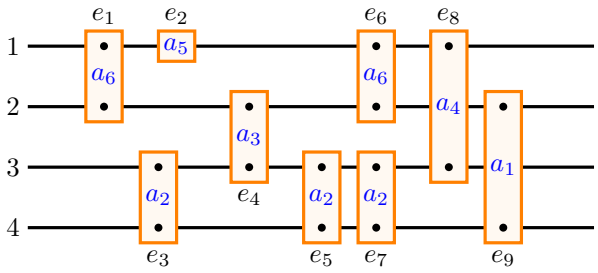


Figure 1 A process line represents the sequence of actions in which the corresponding process participates. Each event is represented by a rectangle with its tied action inside.

▶ **Example 2.** A trace with 9 events over 4 processes with $\mathscr{P} = \{1, 2, 3, 4\}$ is shown in Figure 1 over a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i \in \mathscr{P}}$ where, $\Sigma_1 = \{a_4, a_5, a_6\}$ and $\Sigma_2 = \{a_1, a_3, a_6\}$ and so on. Each process is indicated by a horizontal line and time flows rightward. Each event is represented by a vertical box and is labelled by a letter in $\Sigma = \{a_1, a_2, a_3, a_4, a_5, a_6\}$. Some events such as e_2 are purely local to a single process. Dots in synchronizing events indicate the participating processes. It is easy to infer causality relation ($\leq$) from the figure, for example, $e_1 < e_4 < e_5 < e_7$. We can also infer which events are concurrent (unrelated by $\leq$): for example, event e_6 is concurrent to events e_5, e_7; e_8 and e_9 are also concurrent. The pairs of events e_4, e_5 and e_1, e_4 are causally ordered by process 3 and process 2, respectively.

Let $\mathrm{TR}(\widetilde{\Sigma})$ denote the set of all traces over $\widetilde{\Sigma}$. As the distribution of Σ across processes will be clear from the context, we use $\mathrm{TR}(\Sigma)$ (or simply TR) instead of $\mathrm{TR}(\widetilde{\Sigma})$. Henceforth a trace means a trace over $\widetilde{\Sigma}$. A trace is finite if the underlying set of events is finite. We use $\mathrm{TR}^*(\Sigma)$ and $\mathrm{TR}^\omega(\Sigma)$ (or simply TR^* and TR^ω) to denote the set of all finite and infinite traces. Now we describe the concatenation operation on traces. Let $t = (E, \leq, \lambda)$ and $t' = (E', \leq', \lambda')$ be *finite* traces with disjoint event sets E and E'. We define $tt' \in \mathrm{TR}^*$ to be the trace $(E'', \leq'', \lambda'')$ where

- $E'' = E \cup E'$,
- $\leq''$ is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda'(e')) \in D\}$,
- $\lambda'' \colon E'' \to \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

Observe that, with a (resp. b) denoting the singleton trace whose only event is labelled a (resp. b), if $(a, b) \in I$ then $ab = ba$ in TR^*. The trace concatenation operation on finite traces can also be defined for infinite traces under certain conditions. We explain this now.

The above mentioned definition of tt' results in a valid trace if t is finite. That is, if $t \in \mathrm{TR}^*$ and $t' \in \mathrm{TR}$ then $tt' \in \mathrm{TR}$. Moreover, if $t, t' \in \mathrm{TR}$ are such that $\omega\mathrm{loc}(t) \cap \mathrm{loc}(t') = \emptyset$, then $tt' \in \mathrm{TR}$ where $\omega\mathrm{loc}(t)$ is the set of of processes which participate in infinitely many events in t. It is easy to see that, in this case, $\omega\mathrm{loc}(tt') = \omega\mathrm{loc}(t) \cup \omega\mathrm{loc}(t')$. We say that a trace t' is a *prefix* of a trace t if $t = t't''$ for some trace t''. It is important to observe that this definition permits a prefix to be infinite.

We now come to the very important notion of a configuration of a trace $t = (E, \leq, \lambda)$. A subset $c \subseteq E$ is a *configuration* of t if c is *finite* and $\downarrow c = c$. For $e \in E$, $\downarrow e = \downarrow\{e\}$ is the causal past of e and $\Downarrow e = \downarrow e \setminus \{e\}$ is the strict causal past of e. Note that, if we restrict the trace t to a configuration c, we get another *finite* trace $t_c = (c, \leq, \lambda)$ which turns out to be a *prefix* of t. Conversely, given a finite prefix t' of t, we can identify a configuration c of t such that $t' = t_c$. In this sense, configurations of t are essentially finite prefixes of t. Examples of configurations (or, equivalently, finite prefixes) are the empty set, $\downarrow e$ or $\Downarrow e$, for every event $e \in E$. Note that E itself is a configuration of t iff t is finite. We let C_t denote the set of all configurations of t. The *action based successor relation* $\rightarrow_t \subseteq C_t \times \Sigma \times C_t$ is defined by $c \xrightarrow{a}_t c'$ if and only if there exists $e \in E$ such that $e \notin c$, $c \cup \{e\} = c'$ and $\lambda(e) = a$.

▶ **Example 3.** For the trace t in Figure 1, $c = \{e_1, e_3, e_4\}$ and $c' = \{e_1, e_2, e_3, e_4\}$ are configurations, and so is $c'' = \downarrow e_5 = \{e_1, e_3, e_4, e_5\}$. Observe that $c \xrightarrow{a_5}_t c'$ and $c \xrightarrow{a_2}_t c''$. The set $\{e_1, e_2, e_4\}$ is not a configuration since $\downarrow\{e_1, e_2, e_4\} = \{e_1, e_2, e_3, e_4\}$.

Asynchronous automata and the related transition systems are fundamental finite-state distributed devices due to Zielonka [20, 16] which operate/run on traces. In an asynchronous transition system, each process $i \in \mathscr{P}$ is equipped with a finite non-empty set S_i of *local* i-states. We set $S = \bigcup_{i \in \mathscr{P}} S_i$ and call S the set of local states. Further, we set $S_{\mathscr{P}} = \prod_{i \in \mathscr{P}} S_i$ and call it the set of *global* states. For a non-empty set $P \subseteq \mathscr{P}$ of processes, $S_P = \prod_{i \in P} S_i$ is the set of (joint) P-states. For a P-state $s \in S_P$ and $i \in P$, $s(i) \in S_i$ denotes the local i-state in the P-tuple s. More generally, for $Q \subseteq P$ and $s \in S_P$, we denote by s_Q the projection of s on Q – that is, s_Q is the unique Q-state such that $s_Q(i) = s(i)$ for all $i \in Q$. For $a \in \Sigma$, we use a to abbreviate $\mathrm{loc}(a)$ when talking about states. Thus $S_a = S_{\mathrm{loc}(a)}$ denotes the set of all a-states and if $\mathrm{loc}(a) \subseteq P$ and s is a P-state, we write s_a for $s_{\mathrm{loc}(a)}$.

Similar to the convention of writing a $\mathscr{P}$-indexed family as $\{X_i\}$, we use the shorthand $\{Y_a\}$ to denote the Σ-indexed family $\{Y_a\}_{a \in \Sigma}$.

▶ **Definition 4.** *An asynchronous transition system (ATS) over $\widetilde{\Sigma}$ is $A = (\{S_i\}, \{\xrightarrow{a}\})$ where,*

- *For each process $i \in \mathscr{P}$, S_i is a finite non-empty set of local i-states.*
- *For each action $a \in \Sigma$, $\xrightarrow{a} \subseteq S_a \times S_a$ is a* non-deterministic *transition relation on a-states.*

Let $A = (\{S_i\}, \{\xrightarrow{a}\})$ be an ATS. Note that a transition in A on an action a is *local* in the sense that it involves only processes in $\mathrm{loc}(a)$ and a-states. We extend these local transition relations naturally to global states. More precisely, for $a \in \Sigma$, we define $\overset{a}{\Longrightarrow} \subseteq S_{\mathscr{P}} \times S_{\mathscr{P}}$ as follows: for $s, s' \in S_{\mathscr{P}}$, $(s, s') \in \overset{a}{\Longrightarrow}$ if $(s_a, s'_a) \in \xrightarrow{a}$ and $s_{\mathscr{P} \setminus \mathrm{loc}(a)} = s'_{\mathscr{P} \setminus \mathrm{loc}(a)}$. An action a is said to be *enabled* at $s \in S_{\mathscr{P}}$ if there exists $s' \in S_{\mathscr{P}}$ such that $(s, s') \in \overset{a}{\Longrightarrow}$. Due to the *locality of transitions*, if a is enabled at s, then it is also enabled at s' with $s_a = s'_a$.

Now we define the important notion of a run of A on a trace t over $\widetilde{\Sigma}$. Let us fix an initial global state $s_0 \in S_{\mathscr{P}}$. A run of A on $t \in \mathrm{TR}$, starting from s_0, is a map $\rho : C_t \to S_{\mathscr{P}}$ such that $\rho(\emptyset) = s_0$ and for every $c, c' \in C_t$, $c \xrightarrow{a}_t c'$ implies that $(\rho(c), \rho(c')) \in \overset{a}{\Longrightarrow}$. Note that, as A is non-deterministic, there can be multiple runs of A on the same trace. The following lemma follows directly from the locality of transitions of A.

▶ **Lemma 5.** *Let $t = (E, \leq, \lambda)$ be a trace and $\rho : C_t \to S_{\mathscr{P}}$ be a run of A on t starting at s_0. Then ρ is determined by $\rho(\emptyset)$ and $\rho(\downarrow e)$ for each $e \in E$.*

3 The model

Now we describe the model called **a**synchronous **t**ransition **s**ystem games or simply ATS games. ATS games on two processes are studied in [1]. An ATS game is played on a non-deterministic asynchronous transition system between an *environment* and a *distributed system* that comprises of a *team of cooperating processes*. The environment is a singular entity and manifests itself in the form of a *scheduler* of actions. Importantly, the environment *does not reveal* any information about prior/concurrent scheduling events.

We continue with the notation from the previous section. In particular, we fix a distributed alphabet $\widetilde{\Sigma}$ over a fixed set $\mathscr{P}$ of processes.

▶ **Definition 6.** *An ATS game is of the form* $\mathcal{G} = (A, s_0, \mathrm{Win})$ *where*
- $A = (\{S_i\}, \{\xrightarrow{a}\})$ *is an asynchronous transition system over* $\widetilde{\Sigma}$.
- $s_0 \in S_{\mathscr{P}}$ *is an initial global state of* A.
- Win *is a specification of the winning condition.*

A play of an ATS game is inherently distributed, capturing the ongoing, possibly non-terminating interaction between the distributed system and the environment. Let us now formalize this notion of a *distributed play*. To this end, fix an ATS game $\mathcal{G} = (A, s_0, \mathrm{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$.

We begin with a description of the interleaved semantics of a distributed play. Such a play begins in the initial global state s_0. Environment makes the first move by *scheduling* an action (say a) which is enabled at s_0. The processes participating *in* a respond by selecting an available a-transition and thus advancing the "current" global state (say, to s). At this point, it is environment's turn to schedule another action (say b) which is enabled at s. The processes participating *in* b respond by choosing a suitable b-transition at s. Let us assume that a and b are independent actions. Then the processes participating in b are oblivious to the "concurrent" scheduling of "prior" a-action. It is important to observe that, in this situation, thanks to the locality of transitions in A, b is also enabled at s_0 and environment also had the option of scheduling b first and then a later. On the other hand, if a and b are dependent, then there is a process which participates in both a and b and every process participating in b *comes to know* about the prior a-action which is "causally" before the current b-action.

So, in this interleaved semantics of a distributed play, a play consists of an alternate sequence of moves of the environment (which schedules actions) and the distributed system (where the participating processes respond by matching transitions). Observe that each process in the distributed system has only partial information; among all the actions which are already scheduled, it only *knows* those which are in its *causal past*. The causal past of a process contains the information that a process comes to know directly or indirectly through its interactions with the other processes. Further when two or more processes interact (by participating in a shared/joint action), they exchange their complete causal pasts.

▶ **Definition 7.** *A (partial and distributed) play of the game* $\mathcal{G}$ *is a tuple* (t, ρ) *where*
- $t = (E, \leq, \lambda)$ *is a trace over* $\widetilde{\Sigma}$.
- $\rho : C_t \to S_{\mathscr{P}}$ *is a run of* A *on* t *starting at* s_0

The idea is that the (labelled) events in t represent *enabled* scheduler choices and the partial-order $\leq$ captures the causality between these events arising out of the distribution of these events across processes. The fact that an event e is labelled a (that is, $\lambda(e) = a$) means that the environment scheduled the shared action a which lead the processes participating

in a to synchronize. Their *collective causal past-information* at e corresponds to events in $\Downarrow e$. Based on this common information, these processes respond to e by choosing a local transition on a which advances their prior joint a-state s_a to s'_a where $s = \rho(\Downarrow e)$ and $s' = \rho(\downarrow e)$. Thus the run ρ correctly captures the decisions of the processes during the play. Note that by Lemma 5, ρ is completely determined by $\rho(\emptyset) = s_0$ and $\rho(\downarrow e)$ for each $e \in E$. In summary, the event e models environment's scheduling of action $\lambda(e)$ at the global state $\rho(\Downarrow e)$; and the $\lambda(e)$-transition from $\rho(\Downarrow e)$ to $\rho(\downarrow e)$ models the collective response of the processes participating in e.

A (distributed) play (t, ρ) is said to be *maximal* if there is no play (t', ρ') such that t is a proper prefix of t' and $\rho = \rho'|_{C_t}$, the restriction of ρ' to the configurations of t. Note that if (t, ρ) is maximal then the processes which have moved only finitely often in t can not be further scheduled by the environment as no action involving *only them* is enabled. It is important to note that a maximal play could be either finite or infinite. If a maximal play $(t = (E, \leq, \lambda), \rho)$ is finite, then no action is enabled at the final global state $\rho(E)$.

We next turn our attention to the winning condition Win which specifies the winner of a maximal play. At an abstract level, Win is simply the collection of all maximal plays in which the distributed system wins.

In this work we study ATS games from the viewpoint of *decision making processes*.

▶ **Definition 8.** *Let $\mathcal{G} = (A, s_0, \mathrm{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$. An action a of A is* not *deterministic if there exists $s_a, s'_a, s''_a \in S_a$ such that both (s_a, s'_a) and (s_a, s''_a) are a-transitions (that is, belong to $\xrightarrow{a}$) and $s'_a \neq s''_a$.*

A subset $Q \subset \mathscr{P}$ of processes is said to be decision makers *in $\mathcal{G}$ if for every a which is* not *deterministic, $Q \cap \mathrm{loc}(a) \neq \emptyset$. In other words, if no process in Q participates in an action b, then the transition relation $\xrightarrow{b}$ of A is a (partial) deterministic function from S_b to S_b.*

A **c***entral* **d***ecision* **m***aker (CDM) game is an ATS game with decision makers $\{\ell\}$ for some process $\ell \in \mathscr{P}$. This central decision maker process ℓ participates in* every *action of A which is* not *deterministic.*

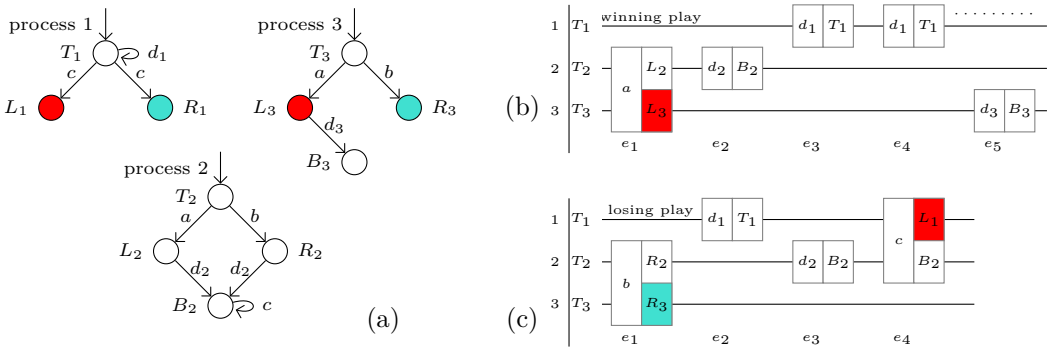


Figure 2 A safety CDM game with unsafe set $\{(L_1, B_2, R_3), (R_1, B_2, L_3)\}$ and its plays.

▶ **Example 9.** Consider a CDM game illustrated in Figure 2 with $\mathscr{P} = \{1, 2, 3\}$ and process 1 as the central decision maker. Note that only action c is *not* deterministic and it is shared by 1 and 2. Process i has purely local action d_i and actions a and b are shared by 2 and 3.

All 3 processes start in top local states and want to avoid states where colors mismatch. This can be captured as a global safety winning condition where color-mismatch signifies occurrence of an unsafe global state. Observe that, initially, the scheduler can play exactly

one of the deterministic actions a or b *forcing* the next joint state of process 2 and process 3 to be either (L_2, L_3) or (R_2, R_3). It can concurrently schedule any number of d_1 actions. Afterwards, the scheduler can *force* process 2 to the bottom local state B_2 by playing local action d_2. At this point, actions c and d_1 are *enabled* and the scheduler is allowed to play any of these actions. Note that, if process 3 is in local state L_3 then the scheduler must also schedule d_3 in a maximal play. Some plays of this CDM game are depicted in Figure 2.

If action c is never scheduled, the distributed system wins along such maximal plays. However, if action c is ever scheduled then the resulting response determines the winner. In order to win in such situations, the central decision maker process 1 needs to inspect its *causal past* to find out which of the actions a or b was played in the past. With this information, process 1 makes the correct decision and ensures a win for the team.

Now we are ready to define the important and crucial notion of a distributed strategy for an ATS game. Intuitively speaking, a distributed strategy is an advice function that the team of processes in the distributed system uses to respond to environment's actions. More importantly, this response can only depend on the collective causal history of the processes participating in this action. Further, the advice function does not restrict the choices available to the environment in all situations.

▶ **Definition 10.** *A distributed strategy in $\mathcal{G}$ is a partial function $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ with the smallest domain such that*

- *the domain of σ is prefix closed, that is, if $t \in \mathrm{TR}^*$ is such that $\sigma(t)$ is defined and $t' \in \mathrm{TR}^*$ is a prefix of t then $\sigma(t')$ is also defined.*
- *$\sigma(\epsilon) = s_0$ where ϵ denotes the empty trace.*
- *for every $t \in \mathrm{TR}^*$ if $\sigma(t)$ is defined and a is enabled at $\sigma(t)$, then $\sigma(ta)$ is also defined, and $(\sigma(t), \sigma(ta)) \in \stackrel{a}{\Longrightarrow}$ (here ta denotes the concatenation of t with the singleton trace a).*

It is easy to see that the first condition in the above definition, that the domain of σ is prefix-closed, is redundant. By requiring σ to have the smallest domain, the second and third conditions in the above definition ensure that σ becomes defined exactly on traces built by iteratively extending the empty trace along enabled actions, making the domain of σ prefix-closed.

A finite trace $t = (E, \leq, \lambda) \in \mathrm{TR}^*$ is said to be *prime* if t has a unique maximum event. In other words, a prime trace t has an event $e \in E$ such that $E = {\downarrow}e$; we let $\mathrm{last}(t)$ denote this maximum event of t. The next lemma states that a distributed strategy is determined by its effect on the prime traces.

▶ **Lemma 11.** *Let $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathcal{G}$. Then σ is completely determined by $\sigma(t)$ for every prime trace t in the domain of σ. Moreover, if $\mathcal{G}$ is a CDM game with ℓ as a central decision maker then σ is completely determined by $\sigma(t)$ for every prime trace t in the domain of σ whose last action $\mathrm{last}(t)$ involves ℓ.*

▶ **Definition 12.** *Let $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathcal{G}$. A distributed play $(t, \rho : C_t \to S_{\mathscr{P}})$ of $\mathcal{G}$ is said to* conform *σ if, for all $c \in C_t$, $\rho(c) = \sigma(c)$. Recall that, for a configuration c of t, c also denotes the induced finite trace $t_c = (c, \leq, \lambda)$.*

A distributed strategy $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ is winning in $\mathcal{G}$ if all maximal plays conforming it belong to Win*. In other words, all maximal plays where the distributed team employs σ are won by the distributed system.*

The description at the end of Example 9, for the CDM game in Figure 2, can be readily converted into a winning distributed strategy as in the above definition.

Given an ATS game $\mathcal{G} = (A, s_0, \text{Win})$, the key algorithmic question is to decide if there exists a distributed winning strategy in $\mathcal{G}$. We answer this for CDM games with global safety and local parity conditions. Note that a CDM game is a partial-information game and different processes have mutually incomparable partial informations about scheduler's actions; a process is oblivious to concurrent scheduling decisions on other processes and there is no bound on the number of such concurrent scheduling decisions. Clearly the decisions taken by the central decision maker (based solely on its causal past) influence the future behaviour of other processes. Observe that ATS games with only one process ($|\mathscr{P}| = 1$) are CDM games and are essentially full-information two-player games played on a finite graph [13].

4 Extraction of distributed strategies via special linearizations

We now develop a general construction which aids in the solution of CDM games. Let us fix a CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and process $1 \in \mathscr{P}$ as the central decision maker. Towards solving $\mathcal{G}$, we introduce a standard *full-information two-player* game G_{seq} played on a finite graph A_{seq} between two players called Sys and Env. See [13] for a study of these games and its relevance to reactive synthesis in the sequential setting.

▶ **Definition 13.** *The game arena A_{seq} is a bipartite graph $A_{\text{seq}} = (V_{\text{env}}, V_{\text{sys}}, \rightsquigarrow \subseteq V_{\text{env}} \times V_{\text{sys}} \cup V_{\text{sys}} \times V_{\text{env}})$ whose vertices and directed edges (that is, elements of $\rightsquigarrow$) are as follows:*

- *$V_{\text{env}} = S_{\mathscr{P}}$ and $V_{\text{sys}} = \{(s, a) \in S_{\mathscr{P}} \times \Sigma \mid a \text{ is enabled at } s\}$*
- *For $s \in V_{\text{env}}, (s', a) \in V_{\text{sys}}$, we have an edge from s to (s', a) iff $s = s'$*
- *For $(s', a) \in V_{\text{sys}}$ and $s \in V_{\text{env}}$, we have an edge from (s', a) to s iff $s' \overset{a}{\Longrightarrow} s$ in the ATS A. Recall that $\overset{a}{\Longrightarrow}$ naturally extends the local transition relation $\xrightarrow{a}$ of A to global-states.*

We define $G_{\text{seq}} = (A_{\text{seq}}, s_0, \text{Win}_{\text{seq}})$ to be a standard two-player graph game of complete information whose winning condition Win_{seq} is currently *unspecified*. Note that $s_0 \in V_{\text{env}}$. The game G_{seq} is played by players Sys and Env by alternately moving a "token" along the edges of the bipartite graph A_{seq}. Player Env makes moves from V_{env} and player Sys makes moves from V_{sys}. Initially the token is at vertex $s_0 \in V_{\text{env}}$. Thus a play of G_{seq} is a sequence (possibly infinite) of vertices visited by the token starting with vertex s_0 and it is of the form: $s_0, (s_0, a_0), s_1, (s_1, a_1), s_2, \ldots$. A maximal play is either infinite, or finite in which case it ends in $s \in V_{\text{env}}$ such that no action is enabled at s and as a result, there is no outgoing-edge in A_{seq} from the vertex s. As mentioned above, we do not specify the winning condition Win_{seq} in this section and it will be suitably instantiated in later sections.

It will be useful to view a play of G_{seq} as simply a sequence $s_0, a_0, s_1, a_1, s_2, \ldots$ starting at s_0 such that, for each i, a_i is enabled at s_i and $(s_i, s_{i+1}) \in \overset{a_i}{\Longrightarrow}$. In this viewpoint, a_i corresponds to player Env moving the token from s_i to (s_i, a_i) and, s_{i+1} corresponds to player Sys moving the token subsequently from (s_i, a_i) to s_{i+1}.

▶ **Example 14.** Figure 3 shows an ATS A and a simplified view of the derived game arena A_{seq}. The simplified view represents the *sequential automaton* underlying A_{seq}. For instance, in the graph A_{seq} we have two edges from TBL (short for $T_1B_2L_3$) to (TBL, c) and (TBL, d_3) corresponding to two enabled actions, namely c and d_3, at TBL. We also have two edges in A_{seq} from (TBL, c) to LBL and RBL corresponding to two global c-transitions $(TBL, LBL), (TBL, RBL)$ of A. A play $s_0, a_0, s_1, a_1, s_2, \ldots$ starting at s_0 in G_{seq} described above, can be simply viewed as a *run* of this sequential automaton on $a_0a_1a_2, \ldots$ from s_0.

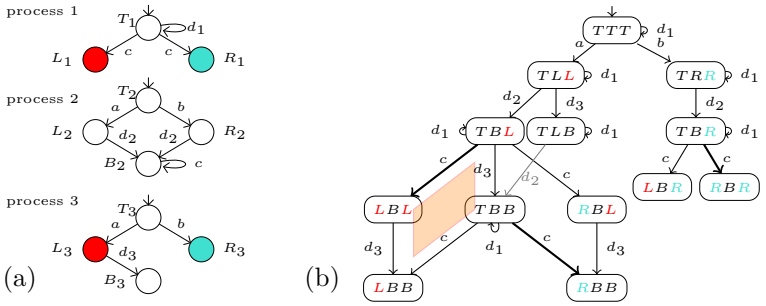


Figure 3 An ATS A and the derived sequential game arena A_{seq}.

A (sequential full-information) strategy for player Sys in G_{seq} maps its history leading to current vertex to the next valid vertex. We formalize it as follows.

▶ **Definition 15.** *A strategy for player* Sys *in* G_{seq} *is a partial function* $\tau : \Sigma^* \to S_{\mathscr{P}}$ *with the smallest domain such that*

- $\tau(\epsilon) = s_0$ *where* ϵ *denotes the empty word.*
- *for every* $w \in \Sigma^*$ *if* $\tau(w)$ *is defined and* a *is enabled at* $\tau(w)$*, then* $\tau(wa)$ *is also defined and furthermore* $(\tau(w), \tau(wa)) \in \stackrel{a}{\Longrightarrow}$*. In other words, we have a directed edge in* A_{seq} *from* $(\tau(w), a) \in V_{\text{Sys}}$ *to* $\tau(wa) \in V_{\text{Env}}$*.*

It is easy to check that the domain of τ *is a prefix-closed subset of* Σ^**.*

Henceforth, by a strategy in G_{seq} we always mean a strategy *for the player* Sys in G_{seq}. It turns out that every distributed strategy in $\mathcal{G}$ naturally gives rise to a sequential strategy in G_{seq}. We develop some more notation to explain this. Each finite word $w \in \Sigma^*$ naturally induces a finite trace $t \in \text{TR}^*$. More precisely, if $w = a_1 a_2 \cdots a_n$, we associate with w the finite trace $t = (E, \leq, \lambda)$ which is defined as follows: $E = \{e_1, e_2, \ldots, e_n\}$ has n events corresponding to n positions in the word w; we set $\lambda(e_i) = a_i$ and the partial order $\leq$ on E is the transitive-closure of the relation $\{(e_i, e_j) | i \leq j \text{ and } (a_i, a_j) \in D\}$. It is easy to check that t is indeed a trace over $\widetilde{\Sigma}$. Let us denote this association by the map $\eta : \Sigma^* \to \text{TR}^*$. It turns out [4, 3, 16] that for a trace $t \in \text{TR}^*$, the set $\eta^{-1}(t) \subseteq \Sigma^*$ is precisely all those words which correspond to different linearizations of t. In fact, η is a surjective monoid morphism and $\eta(w) = \eta(w')$ iff w can be obtained from w' by a sequence of operations where each operation is an *exchange* of two *adjacent independent* letters.

The next proposition states that, a distributed strategy σ in $\mathcal{G}$ naturally induces a (sequential) strategy σ_{seq} in G_{seq}.

▶ **Proposition 16.** *Let* $\sigma : \text{TR}^* \to S_{\mathscr{P}}$ *be a distributed strategy in* $\mathcal{G}$*. Consider the induced partial* function $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ *defined as follows: for* $w \in \Sigma^*$*,* $\sigma_{\text{seq}}(w) = \sigma(\eta(w))$*. So* w *belongs to the domain of* σ_{seq} *iff* $\eta(w)$ *belongs to the domain of* σ*. Then* $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ *is in fact a strategy for player* Sys *in* G_{seq}*. Furthermore,* σ_{seq} *is* diamond-closed*; that is, for* $w, w' \in \Sigma^*$ *and* $(a, b) \in I$*,* $\sigma_{\text{seq}}(wabw') = \sigma_{\text{seq}}(wbaw')$*.*

So, the strategies in G_{seq} induced by distributed strategies in $\mathcal{G}$ are diamond-closed. However, an arbitrary strategy in G_{seq} is not necessarily diamond-closed.

▶ **Example 17.** In Figure 3(b), we have used **bold** edges to depict the response of a *positional* strategy in G_{seq} on the action c which is not deterministic. This strategy responds to c at TBL by using the bold edge to LBL, at TBB by using the bold edge to RBB and at TBR

by using the bold edge to RBR. As a result, at TBL the response to cd_3 ends at LBB while that for d_3c ends at RBB. As c and d_3 are independent, this strategy is not diamond-closed.

Our main goal in this section is to extract a distributed strategy in $\mathcal{G}$ from an arbitrary strategy crucially exploiting the presence of a central decision maker. Towards this, we now develop special linearizations of finite traces.

We first fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. Note that, in the total order $\lhd_\Sigma$, if $a, b \in \Sigma$ are such that $1 \notin \mathrm{loc}(a)$ and $1 \in \mathrm{loc}(b)$ then $a \lhd_\Sigma b$. With our implicit assumption that process 1 is the central decision maker, every action in which it does not participate comes before every action in which it participates in the order $\lhd_\Sigma$. We now outline an inductive procedure to compute a special linearization map $\mathrm{Lin} : \mathrm{TR}^* \to \Sigma^*$.

▶ **Definition 18.** *Let $t = (E, \leq, \lambda) \in \mathrm{TR}^*$. If $t = \epsilon$, then $\mathrm{Lin}(t) = \epsilon$. Otherwise, let M be the set of all maximal (wrt $\leq$) events of the trace t and e be the unique event in M whose label is $\lhd_\Sigma$-least among the labels in $\{\lambda(f) \mid f \in M\}$. Note that no two events in M can have the same label as events with the same label are ordered/comparable wrt $\leq$.*

It is easily verified that with t' as the trace corresponding to the configuration $E \setminus \{e\}$, we have $t = t'a$ where $a = \lambda(e)$. We now define $\mathrm{Lin}(t) = \mathrm{Lin}(t')a$.

In short, we compute $\mathrm{Lin}(t)$ starting from its last action/letter. Towards this, we peel off that *maximal* event of t which is $\lhd_\Sigma$-least labelled and put its label as the *last* letter in $\mathrm{Lin}(t)$ and continue this process until all events are covered.

Now we exploit the key consequences of the $\lhd_\Sigma$ requirement that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. It is very important to observe that, in general, t' is a trace-prefix of t *does not* imply that $\mathrm{Lin}(t')$ is a word-prefix of $\mathrm{Lin}(t)$. The following lemma plays a crucial role later.

▶ **Lemma 19.** *Let $t = (E, \leq, \lambda) \in \mathrm{TR}^*$ and $e \in E$ be such that process 1 participates in e, that is $\lambda(e) \in \Sigma_1$. Then $\mathrm{Lin}(\downarrow e)$ is prefix of $\mathrm{Lin}(t)$. Here we identify the configuration $\downarrow e$ with the corresponding trace-prefix $(\downarrow e, \leq, \lambda)$ of t.*

▶ **Example 20.** We revisit the trace t from Figure 1 with process 1 as the central decision maker and event set $E = \{e_1, e_2, \ldots, e_9\}$. Let the ordering $\lhd_\Sigma$ be given by $a_i \lhd_\Sigma a_j$ iff $i \leq j$. Note that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. We now compute $\mathrm{Lin}(t)$ for the trace t. Maximal events of t are e_8 and e_9 and as $\lambda(e_9) = a_1 \lhd_\Sigma a_4 = \lambda(e_8)$, $\mathrm{Lin}(t) = \mathrm{Lin}(t').a_1$ where t' is given by $E' = E \setminus \{e_9\}$. Successively the maximal event sets are $\{e_8\}, \{e_6, e_7\}, \{e_6, e_5\}, \{e_6\}, \{e_2, e_4\}, \{e_2, e_3\}, \{e_2\}, \{e_1\}$. This results in $\mathrm{Lin}(t) = a_6a_5a_2a_3a_6a_2a_2a_4a_1$ with event-linearization $e_1e_2e_3e_4e_6e_5e_7e_8e_9$.

As an illustration of Lemma 19, consider event $e_6 \in E_1$. We have $\downarrow e_6 = \{e_1, e_2, e_3, e_4, e_6\}$ and $\mathrm{Lin}(\downarrow e_6) = a_6a_5a_2a_3a_6$ which is a prefix of $\mathrm{Lin}(t)$. However, $e_7 \notin E_1$ and one can verify that $\mathrm{Lin}(\downarrow e_7) = a_6a_5a_2a_3a_2a_2$, which is *not* a prefix of $\mathrm{Lin}(t)$.

Now we use special linearizations to lift a strategy in G_{seq} to a distributed strategy in $\mathcal{G}$. Intuitively, in the special linearization of a trace, all the events of process 1 appear the *earliest*. Thus the response of a sequential strategy along the special linearization is based on the least amount of information to the central decision maker. We lift these responses to construct a distributed strategy.

▶ **Definition 21.** *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a strategy in G_{seq}. We define a partial function $\tau_{\mathrm{dstr}} : \mathrm{TR}^* \to S_{\mathscr{P}}$ as follows: for $t \in \mathrm{TR}^*$, $\tau_{\mathrm{dstr}}(t) = \tau(\mathrm{Lin}(t))$. Note that, t belongs to the domain of τ_{dstr} iff $\mathrm{Lin}(t)$ belongs to the domain of τ.*

Let us illustrate the above definition for the non-diamond-closed strategy τ from Example 17 with t being the trace induced by the word ad_2d_3c. As $\mathrm{Lin}(t) = ad_2cd_3$, we set $\tau_{\mathrm{dstr}}(t) = \tau(ad_2cd_3) = LBB$. Note that $\tau(ad_2d_3c) = RBB$.

We now state the following main proposition which shows that *partial function* τ_{dstr} from Definition 21 is in fact a distributed strategy in $\mathcal{G}$. The proof crucially uses Lemma 19 which implies that, restricted to central decision maker events, special linearization of a later event *extends* that of an earlier one.

▶ **Proposition 22.** *Let* $\tau : \Sigma^* \to S_{\mathscr{P}}$ *be a strategy for player* Sys *in* G_{seq}. *Then* $\tau_{\text{dstr}} : \text{TR}^* \to S_{\mathscr{P}}$ *is a distributed strategy in* $\mathcal{G}$.

5 Global safety and local parity CDM games

We turn our attention to CDM games with global safety and local parity objectives. For an ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$, $\max_i |S_i|$ denotes the maximum number of local states per process. For complexity analysis, we make the realistic assumption that every action involves at most a *fixed constant* number of participating processes. This is reasonable because in most distributed systems, each action involves only a few processes interacting at a time, e.g., reading/writing shared resources or coordinating with immediate "neighbors". Thanks to this assumption and the local nature of transitions of A, it is easy to check that the size of A, denoted by $||A||$, is *polynomial* in $\max_i |S_i|$, $|\Sigma|$ and $|\mathscr{P}|$. Besides $||A||$, the other component which contributes to the size of a CDM game $\mathcal{G}$, denoted $||\mathcal{G}||$, is the specification of the winning condition. Observe that the number of vertices in the game arena A_{seq} (see Definition 13) is atmost $2 * |\Sigma| * (\max_i |S_i|)^{|\mathscr{P}|}$. Note that $(\max_i |S_i|)^{|\mathscr{P}|}$ is the maximum number of global-states of A. As a result, the size of A_{seq}, denoted by $||A_{\text{seq}}||$ is polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}$ and $|\Sigma|$.

5.1 Global safety objective

Let us fix a CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and a global-safety winning condition. A global-safety winning condition is given by a subset $F \subseteq S_{\mathscr{P}}$ of *safe* global-states. A maximal play (t, ρ) is won by the distributed system if, for *all* $c \in C_t$, $\rho(c) \in F$. By abuse of notation, we also write $\mathcal{G} = (A, s_0, F)$ and refer to it as a safety CDM game.

Our solution for the safety CDM game uses the full information two-player token game $G_{\text{seq}} = (A_{\text{seq}}, s_0, \text{Win}_{\text{seq}})$ from the previous section. We instantiate the winning condition Win_{seq} and the resulting G_{seq} as follows:

▶ **Definition 23.** *Let* $G_{\text{seq}} = (A_{\text{seq}}, s_0, F_{\text{seq}})$ *be the full-information two-player safety graph game where the safety objective for player* Sys *is given by*

$$F_{\text{seq}} = \{s \in V_{\text{sys}} \mid s \in F\} \cup \{(s, a) \in V_{\text{env}} \mid s \in F\}$$

In order to win G_{seq}, *player* Sys *must have a strategy to ensure that the token never leaves the safe-set* F_{seq} *of vertices of* A_{seq}.

Recall that a strategy in G_{seq} (winning or not) always means a strategy *for player* Sys.

▶ **Definition 24.** *Let* $\tau : \Sigma^* \to S_{\mathscr{P}}$ *be a strategy in* G_{seq}. *A play* $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ *of* G_{seq} conforms τ *if, for all* i, $\tau((a_0 a_1 \cdots a_i)) = s_{i+1}$. *The fact that* τ *is a strategy ensures that we have a directed edge in* A_{seq} *from* (s_i, a_i) *to* s_{i+1}. *The strategy* τ *is* winning *if all* maximal *plays* $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ *conforming it have the property that, for all* i, $s_i \in F_{\text{seq}}$.

▶ **Theorem 25.** *There is a distributed winning strategy in the safety CDM game* $\mathcal{G}$ *iff there is a winning strategy in* G_{seq}. *Moreover, it can be decided in time polynomial (more*

accurately, quadratic) in the size of A_{seq} *whether player* Sys *has a winning strategy in* G_{seq}*. The running time complexity of the resulting decision procedure for* $\mathcal{G}$ *is polynomial in* $(\max_i |S_i|)^{|\mathscr{P}|}, |\Sigma|, |F|$.

We consider the decision problem of determining whether a CDM game with global safety objectives admits a distributed winning strategy; the procedure above is EXPTIME due to the exponential dependence on $|\mathscr{P}|$. These games are also EXPTIME-hard. The proof is similar to the corresponding result for the Petri game model with one system token studied in [7]. Together these results imply the following theorem.

▶ **Theorem 26.** *Global safety CDM games are EXPTIME-complete.*

5.2 Local parity objective

Now we analyze CDM games with local parity objectives. Parity winning conditions are central to the theory of two-player graph games and are studied extensively in literature [13]. A standard parity game on a graph of size n and m colors can be solved in time $n^{m+O(1)}$ [15]. A recent advance from [2] brings it down to $n^{\log(m)+6}$ - a quasi-polynomial bound.

A local-parity winning condition is given by a color function $\chi : S_1 \to \{0, 1, \ldots, k\}$ that assigns each CDM local state a color from the finite color set $C = \{0, 1, \ldots, k\}$. Given a maximal play (t, ρ) with $t = (E, \leq, \lambda)$, we define $\inf(t, \rho) = \{s \in S_1 \mid \exists^{\infty} e \in E_1, \rho(\downarrow e)(1) = s\}$. Note that if E_1 - the set of events in which process 1 participates, is finite then $\inf(t, \rho) = \emptyset$. The system wins if $\inf(t, \rho)$ is empty or $\max\{\chi(s)_{s \in \inf(\rho)}\}$ is even. We also denote such a game by $\mathcal{G} = (A, s_0, \chi)$ and refer to it as a (local) parity CDM game. Towards solving $\mathcal{G}$, we instantiate G_{seq} by an appropriate *sequential* parity condition.

▶ **Definition 27.** *Let* $G_{\text{seq}} = (A_{\text{seq}}, s_0, \chi_{\text{seq}})$ *be the standard two-player parity game where* $\chi_{\text{seq}} : V_{\text{sys}} \cup V_{\text{env}} \to \{0, 1, \ldots, k\}$ *is defined as:* $\chi_{\text{seq}}(s) = 0$ *and*

$$\begin{aligned} \chi_{\text{seq}}((s, a)) &= \chi(s(1)) \text{ if } a \in \Sigma_1 \\ &= 0 \text{ if } a \notin \Sigma_1 \end{aligned}$$

We now describe the winning condition Win_{seq} of G_{seq} using χ_{seq}. As discussed in Section 4, a maximal play of G_{seq} may be viewed as a sequence $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ and it corresponds to the maximal movement α' of the token along the path $s_0, (s_0, a_0), s_1, (s_1, a_1), s_2, \ldots$ in A_{seq}. On this maximal sequence α, player Sys wins if it is finite or the highest color occurring infinitely often in $\chi_{\text{seq}}^{\alpha'} = \chi_{\text{seq}}(s_0), \chi_{\text{seq}}((s_0, a_0)), \chi_{\text{seq}}(s_1), \chi_{\text{seq}}((s_1, a_1)), \chi_{\text{seq}}(s_2), \ldots$ is even. The following lemma brings out the connection between χ_{seq} and χ.

▶ **Lemma 28.** *If* α *is finite then player* Sys *wins in* α*. We now assume that* α *is infinite. If actions from* Σ_1 *occur finitely often in* α *then only color* 0 *occurs infinitely often in* $\chi_{\text{seq}}^{\alpha'}$ *and player* Sys *wins in* α*. If actions from* Σ_1 *occur infinitely often in* α*, consider the* infinite *subsequence* β' *obtained from* α' *by restricting it to vertices in* $V_{\text{sys}} \cap (S_{\mathscr{P}} \times \Sigma_1)$*. Suppose* $\beta' = (s_{i_1}, a_{i_1}), (s_{i_2}, a_{i_2}), \ldots$*. Then the highest color occurring infinitely often in* $\chi_{\text{seq}}^{\beta'} = \chi_{\text{seq}}((s_{i_1}, a_{i_1})), \chi_{\text{seq}}((s_{i_2}, a_{i_2})), \ldots$ *is same as that of* $\chi_{\text{seq}}^{\alpha'}$*. Furthermore it is equal to the highest color occurring infinitely often in the sequence* $\chi(s_{i_1}(1)), \chi(s_{i_2}(1)), \ldots$.

Now we state our main result for local parity CDM games.

▶ **Theorem 29.** *There is a distributed winning strategy in the local parity CDM game* $\mathcal{G}$ *iff the player* Sys *has a winning strategy in* G_{seq}.

Given a local parity CDM game $\mathcal{G}$, the decision problem is to determine whether the distributed system has a winning strategy. Thanks to the assumption that each action involves at most a fixed constant number of processes, the size $||\mathcal{G}||$ is polynomial in $\max_i |S_i|$, $|\Sigma|$, $|\mathscr{P}|$, and $|C|$, where C is the color set. The derived standard parity game G_{seq} has arena size polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}$ and $|\Sigma|$ and it uses only $|C|$ colors. Known algorithms for parity games on arenas of size n with m colors run in time $n^{m+O(1)}$ [15] or $n^{\log(m)+6}$ [2], and applying them to G_{seq} gives an EXPTIME procedure for $\mathcal{G}$. Moreover, we can also show that local parity CDM games are EXPTIME-hard even with two colors, yielding the following theorem.

▶ **Theorem 30.** *Local parity CDM games are EXPTIME-complete.*

6 Finite-state distributed strategies

In this section we investigate distributed strategies in CDM games which admit a finite-state implementation. Let $\mathcal{G} = (A, s_0, \text{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$.

▶ **Definition 31.** *A* memory automaton *for $\mathcal{G}$ is a* deterministic *asynchronous automaton $\mathcal{A} = (\{S_i \times M_i\}, \{\delta_a : S_a \times M_a \to S_a \times M_a\}, (s_0, m_0))$ over $\widetilde{\Sigma}$ such that*

- *For $a \in \Sigma$, $(s_a, m_a) \in S_a \times M_a = \Pi_{i \in \text{loc}(a)}(S_i \times M_i)$, if a is enabled at s_a in A then a is also enabled at (s_a, m_a) in $\mathcal{A}$. Further, with $\delta_a((s_a, m_a)) = (s'_a, m'_a)$, $(s_a, s'_a) \in \xrightarrow{a}$; that is, there is an a-transition from s_a to s'_a in A.*
- *$(s_0, m_0) \in S_{\mathscr{P}} \times M_{\mathscr{P}}$ is a global state of $\mathcal{A}$ where s_0 is the initial global state of A.*

In the memory automaton $\mathcal{A}$, process i has access to local memory states M_i. The memory automaton is a distributed automaton which starts in the initial global state (s_0, m_0) and provides a deterministic response to *every* enabled scheduler action using the current joint memory-state. Further, it also updates the joint memory-state deterministically.

A memory automation is *zero-memory* if for each i, M_i is a singleton set. A zero-memory automaton may be viewed as a deterministic *asynchronous sub-automaton* of A.

▶ **Definition 32.** *Let $\mathcal{A}$ be a memory automaton for $\mathcal{G}$ and $\sigma : \text{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathcal{G}$. We say that σ is* realized by *$\mathcal{A}$ if, for every trace t in the domain of σ, $\sigma(t) = s$ where $\mathcal{A}(t) = (s, m)$ – here $\mathcal{A}(t)$ is the final global state attained on the unique run of $\mathcal{A}$ on t.*

A distributed strategy σ in $\mathcal{G}$ is said to be finite-state *if there exists $\mathcal{A}$ which realizes it. Thus the responses of σ are determined by the finite-state device $\mathcal{A}$. A distributed strategy σ realized by a zero-memory automaton is called a* zero-memory distributed strategy*.*

It is well-known (see [13]) that for standard two-player full-information sequential games with safety and parity objectives, existence of a winning strategy implies existence of a positional/"zero-memory" winning strategy. Unfortunately, this is not so for CDM games. It is easy to verify that the safety CDM game from Figure 2 does *not* admit a *zero-memory distributed* winning strategy. However, we have already exhibited a valid distributed winning strategy for this game in Example 9.

Let us fix a safety/parity CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$. Recall that in Sections 5.1 and 5.2, we have shown that we can extract a distributed winning strategy τ_{dstr} in $\mathcal{G}$ from a sequential winning strategy τ in G_{seq}. The key property shared by G_{seq} in both instantiations is that: if player Sys has a winning strategy in G_{seq} then it also has a positional winning strategy. Our main goal now is to obtain a memory automaton realizing the distributed strategy τ_{dstr} in $\mathcal{G}$ which is extracted from a positional strategy τ in G_{seq}.

▶ **Definition 33.** *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a strategy in G_{seq}. We call τ positional if there exists a function $f_\tau : V_{\text{sys}} \to V_{\text{env}}$ such that, for every w in the domain of τ and for every a which is enabled at $\tau(w)$, $\tau(wa) = f_\tau((\tau(w), a))$. We refer to f_τ as the witness function of τ.*

▶ **Proposition 34.** *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a positional strategy in G_{seq} with $f_\tau : V_{\text{sys}} \to V_{\text{env}}$ as the witness function of τ. Further, let $\tau_{\text{dstr}} : \text{TR}^* \to S_{\mathscr{P}}$ be the "extracted" distributed strategy in $\mathcal{G}$ defined as: for $t \in \text{TR}^*$, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$. Then, for a prime trace t of the form $t = t'a$, $\tau_{\text{dstr}}(t) = f_\tau((\tau_{\text{dstr}}(t'), a))$.*

The above proposition shows that the response of τ_{dstr} at the last a-event e of a prime trace $t = t'a$ can be determined (using f_τ) by the processes participating in e provided they can compute $\tau_{\text{dstr}}(t')$. Note that $t' = \Downarrow e$ is their collective causal past at e. Hence $\tau_{\text{dstr}}(\Downarrow e)$ represents the *last/latest global-state* about the entire system that they are aware of at e. This suggests that, in order to realize a finite-state implementation of τ_{dstr}, each process should keep track of the latest global-state that it is aware of, in its local memory. When a subset of processes synchronize on a shared action, they need a *finite-state* mechanism to compute the best global-state that they are *collectively* aware of. It turns out that this can be achieved with the help of the *gossip automaton* from [17].

We first develop some more notation. Let $t = (E, \leq, \lambda)$ be a finite trace, $i \in \mathscr{P}$ and $P \subseteq \mathscr{P}$. Recall that E_i is the set of events in which process i participates. We define $\partial_i(t)$ to be the trace induced by $\downarrow E_i$ – the events in the causal past of process i. Similarly, $\partial_P(t)$ is the trace induced by $\cup_{j \in P} \downarrow E_j$ and it represents the collective causal past of processes in P. We define function $\mathsf{latest}_Q : \text{TR}^*(\Sigma) \times \mathscr{P} \to 2^Q$ that gives which processes in the set Q have the latest information about a given process k in a given trace t. For $Q = \{i_1, i_2, \ldots, i_l\}$, $j \in \mathsf{latest}_Q(t, k)$ iff $\partial_k(\partial_Q(t)) = \partial_k(\partial_j(t))$. We now state the key result from [17].

▶ **Theorem 35.** *There exists an effectively constructible deterministic complete asynchronous automaton $A_{\text{gossip}} = (\{V_i\}, \{\nabla_a\}, v_0)$ over $\widetilde{\Sigma}$, called the gossip automaton, and, for each $Q = \{i_1, i_2, \ldots, i_l\} \subseteq \mathscr{P}$ there exists an effectively computable function $gossip_Q : V_{i_1} \times \ldots \times V_{i_l} \times \mathscr{P} \to 2^Q$ such that, for every trace t and every process k, $\mathsf{latest}_Q(t, k) = gossip_Q(v_Q, k)$, where v is the global state of A_{gossip} reached on the unique run of A_{gossip} on t.*

We now describe the memory automaton $\mathcal{A}$ which realizes τ_{dstr} from Proposition 34. We set $\mathcal{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ where for each i, $M_i = S_{\mathscr{P}} \times V_i$. So, a local memory i-state $m_i = (q_i, v_i) \in M_i$ of process i contains a local gossip i-state v_i and a global state $q_i \in S_{\mathscr{P}}$ in A. Intuitively, q_i is the best global state that process i is aware of. The initial global memory state $m_0 \in M_{\mathscr{P}}$ is defined as: for each $i \in \mathscr{P}$, $m_0(i) = (s_0, v_0(i))$. Note that s_0 (resp. v_0) is the initial global state of A (resp. A_{gossip}).

We now turn our attention to the definition of the transition function δ_a of $\mathcal{A}$. Let us suppose that $\text{loc}(a) = Q = \{i_1, i_2, \ldots, i_l\}$. Fix $(s_a, m_a) \in S_a \times M_a$ such that a is enabled at s_a in A. For each $i \in \text{loc}(a)$, $m_a(i) = (q_i, v_i) \in S_{\mathscr{P}} \times V_i$. Recall that ∇_a is the a-transition function of A_{gossip} and let $v_a \in V_a$ be such that, for each $i \in \text{loc}(a)$, $v_a(i) = v_i$ and $v'_a = \nabla_a(v_a)$. We now define $\delta_a((s_a, m_a))$ as follows. We first compute the best global state $\hat{s} \in S_{\mathscr{P}}$ that processes in $Q = \text{loc}(a)$ are aware of. For each $k \in \mathscr{P}$, we set

$$\hat{s}(k) = q_j(k) \text{ where } j \in Q \text{ is such that } j \in \text{gossip}_Q((v_{i_1}, v_{i_2}, ..., v_{i_l}), k)$$

We next compute $\hat{s}' = f_\tau((\hat{s}, a))$ and define $\delta_a((s_a, m_a))$ to be $(\hat{s}'_a, m'_a)$ where, for each $i \in \text{loc}(a)$, $m'_a(i) = (\hat{s}', v'_a(i))$.

▶ **Theorem 36.** *The distributed strategy τ_{dstr} extracted from the positional sequential strategy τ is finite-state. In fact, it is realized by the memory automaton $\mathcal{A}$ defined above.*

Proof idea. We prove by induction on the size of the trace t that if $\mathcal{A}(t) = (s, m)$ with $m = (q, v)$, then $q_i = \tau_{\text{dstr}}(\partial_i(t))$ and $s = \tau_{\text{dstr}}(t)$. The base case is immediate.

For the induction step, let t get extended by a. By induction, q_i correctly tracks τ_{dstr} on the views of each process. Since τ is memoryless, $\tau_{\text{dstr}}(ta)_a = f_\tau(\tau_{\text{dstr}}(\partial_a(t)), a)_a$. Using the gossip construction, let $j = \text{gossip}_a(v_a, k)$. Then, by Theorem 35, $j = \mathsf{latest}_a(t, k)$ and $\partial_k(\partial_a(t)) = \partial_k(\partial_j(t))$. Thus, the intermediate state $\hat{s}$ satisfies $\hat{s}(k) = \tau_{\text{dstr}}(\partial_k(\partial_a(t)))_k$ for each k, hence $\hat{s} = \tau_{\text{dstr}}(\partial_a(t))$. Updating $\mathcal{A}$ along a then correctly sets $q'_i = \tau_{\text{dstr}}(\partial_i(ta))$ and $s' = \tau_{\text{dstr}}(ta)$ for all i, completing the induction. ◀

When applied to safety/parity CDM games, the above theorem provides a memory automaton realizing a distributed winning strategy. Note that A_{gossip} does not depend on the specification of the game $\mathcal{G}$ at all. *Distributed memory complexity* $\max_i |M_i|$ of this memory automaton is bounded by $(\max_i |S_i|)^{|\mathscr{P}|}.(\max_i |V_i|)$. This exponential dependence on $\mathscr{P}$ is due to the fact that each process stores in its local memory state a *global-state* of A. Our next result shows that this exponential dependence on $\mathscr{P}$ is necessary. Its proof is based on an adaptation of the memory lower bound argument for *generalized reachability* in sequential games from [5].

▶ **Theorem 37.** *For every n, there exists a global safety CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ with $|\mathscr{P}| = n + 1$ such that the central decision maker process has $O(n)$ local states and every other process has $O(1)$ local states. Further, there exists a distributed winning strategy in $\mathcal{G}$. However, any memory automaton realizing a winning distributed strategy in $\mathcal{G}$ must have at least 2^n local memory states for the central decision maker.*

7 Discussion

For CDM games – where a single decision maker process participates in all non-deterministic choices – we have established two main results: (i) the problems of deciding the existence of winning strategies for global safety and local parity objectives are EXPTIME-complete, and (ii) constructions of distributed finite-state implementations of these strategies (when they exist) with optimal memory structures.

It is natural to ask about the decidability status of ATS games in the presence of multiple decision making processes. Towards this, we establish the following theorem.

▶ **Theorem 38.** *ATS games with* two *decision makers are undecidable.*

Our proof of the above theorem uses the key ideas from [12] which showed that six-process asynchronous control games are undecidable. More precisely, we use the intermediate problem *infinite bipartite coloring* introduced in [12] for our reduction. We then revisit the ideas in the proof of the undecidability of six-process asynchronous control games and adapt them to the setting of fourteen-process safety ATS games with only two decision making processes. The undecidability proof relies crucially on the ability of the two decision makers to take truly *concurrent* decisions.

Interestingly, if decision events are causally ordered, even across disjoint sets of processes, the arguments presented in this work can be adapted to recover decidability; we presented them in the CDM setting for convenience. It would be interesting to generalize the results in the CDM setting to arbitrary ω-regular objectives. Identifying natural decidable extensions of the CDM games is also a potential future direction.

References

1 Bharat Adsul and Nehul Jain. Asynchronous transition system games for two processes and their analysis. In *11th Indian Conference on Logic and Its Applications (ICLA 2025)*, volume 15402 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2025. doi:10.1007/978-3-031-89610-1_5.

2 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022.

3 Volker Diekert and Yves Métivier. Partial commutation and traces. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 457–533. Springer, 1997. doi:10.1007/978-3-642-59126-6_8.

4 Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific Publishing Co., Inc., USA, 1995.

5 Nathanaël Fijalkow and Florian Horn. The surprizing complexity of generalized reachability games, 2012. arXiv:1010.2420.

6 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, 2022. doi:10.4230/LIPICS.CSL.2022.20.

7 Bernd Finkbeiner and Paul Gölz. Synthesis in Distributed Environments. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:14, 2018. doi:10.4230/LIPICS.FSTTCS.2017.28.

8 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation*, 253:181–203, 2017. doi:10.1016/J.IC.2016.07.006.

9 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

10 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

11 Hugo Gimbert. On the Control of Asynchronous Automata. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, 2018. doi:10.4230/LIPICS.FSTTCS.2017.30.

12 Hugo Gimbert. Distributed asynchronous games with causal memory are undecidable. *Logical Methods in Computer Science*, Volume 18, Issue 3, September 2022. doi:10.46298/lmcs-18(3:30)2022.

13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

14 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International*

Conference, Hyderabad, India, December 15-18, 2005, Proceedings, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005. doi:10.1007/11590156_16.

15 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.

16 Madhavan Mukund. Automata on distributed alphabets. In *Modern Applications of Automata Theory*, IISc Research Monographs Series 2, pages 257–288. World Scientific, 2012. doi:10.1142/9789814271059_0009.

17 Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Computing*, 10(3):137–148, 1997. doi:10.1007/s004460050031.

18 Anca Muscholl and Igor Walukiewicz. Distributed Synthesis for Acyclic Architectures. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 639–651, 2014. doi:10.4230/LIPICS.FSTTCS.2014.639.

19 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

20 Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO-Theoretical Informatics and Applications*, 21(2):99–135, 1987. doi:10.1051/ITA/1987210200991.

A Correct by Construction Fault Tolerant Voter for Input Selection of a Control System

Arif Ali AP
Department of Computer Science and Engineering, Indian Institute of Technology Palakkad, India

Jasine Babu
Department of Computer Science and Engineering, Indian Institute of Technology Palakkad, India

Deepa Sara John
ISRO Inertial Systems Unit, Indian Space Research Organization, Thiruvananthapuram, India

Abstract

Safety-critical systems use redundant input units to improve their reliability and fault tolerance. A voting logic is then used to select a reliable input from the redundant sources. A fault detection and isolation rules help in selecting input units that can participate in voting. This work deals with the formal requirement formulation, design, verification and synthesis of a generic voting unit for an N-modular redundant measurement system used for control applications in avionics systems. The work follows a correct-by-construction approach, using the Rocq theorem prover.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automated reasoning

Keywords and phrases Fault Tolerant System Design, Formal Verification, Correct by Construction, Input Selection, Interactive Theorem Proving

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.6

Related Version *Full Version*: `https://arxiv.org/abs/2509.22236` [1]

Supplementary Material *Software (Source Code)*: `https://github.com/arifali-ap/Fault_tolerant_voter.git`, archived at `swh:1:dir:7ba9568b1f07d7a7efb9870a53eabd213a91354c`

Funding A part of this work is funded by Indian Space Research Organisation (ISRO) RESPOND project RES-IISU-2022-013 titled 'Formal Analysis and Verification of Redundancy Management Logic for Navigation Processor used in Man-rated Launch Vehicle'.

Acknowledgements The authors acknowledge Saarang S and Mis Ab VP, former undergraduate students of IIT Palakkad, for their involvement in preliminary experiments with the Coq implementation of the voter, and Sandra S, visiting student at IIT Palakkad, for testing support. The authors also thank Murali Krishnan, Professor, National Institute of Technology Calicut, for his useful feedback on the initial draft.

1 Introduction

Redundancy management is used in safety-critical systems to improve their reliability and fault tolerance [10, 11, 7]. Examples of some well known domains of safety critical applications include avionics control, automated vehicles and nuclear reactor control systems [16, 15]. Since the reliability guarantee provided by testing is insufficient, such systems are suitable candidates for formal verification [5]. From the late 1980s, usage of formal verification for the design of fault-tolerant systems [2] was supported by agencies such as NASA, which work in the domain of safety-critical applications [33, 10, 30]. Owre et al. [25] mention the formal verification of fault-tolerant architectures as a major motivation for the development of the PVS theorem prover. Interactive consistency and convergence [21], clock synchronization [31, 23], and redundancy management [10] are some well-studied subproblems


© Arif Ali AP, Jasine Babu, and Deepa Sara John;
licensed under Creative Commons License CC-BY 4.0

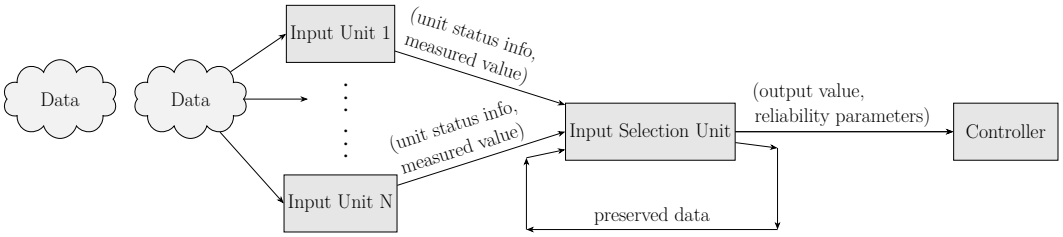


Figure 1 In each cycle, the input selection unit has to select an appropriate measured value and feed it to the controller, along with some parameters indicating the reliability of the output. Between consecutive cycles, the change in the physical quantity being measured is assumed to be small. Preserved data stores some information from past cycles and it is used for fault handling and output generation. The output value could be retained from the previous cycle, when the current cycle measurement is not reliable.

of fault-tolerant system design for which formal verification has been attempted. There are also works on the verification of timing specifications of fault-tolerant systems [29, 32, 14, 34]. Most of the works on formal verification of fault tolerant systems [21, 31, 6, 26, 18, 19] discuss only verification of the formal model and not the code synthesis. There is a recent work by Rahli et al. [28] that discusses Rocq based synthesis of a code implementing a Byzantine fault tolerant state machine replication protocol based on message passing.

Our work deals with the formal modelling, verification and construction of an input selection unit for a synchronous N-modular redundant measurement system used for control applications in avionics systems. Though the redundant input units work synchronously and measure the same physical quantity, there can be differences in the measured value due to the noises in the measurement system (see Section 2.2). Moreover, some of these input units may have transient faults due to factors such as cosmic radiations [24] and vibrations [12], or permanent faults due to hardware failures [8]. Hence, it becomes necessary to apply an input selection logic to the values obtained from the redundant input units and produce a single output value corresponding to the physical quantity. In this work, we follow a correct-by-construction approach for synthesis of the input selection unit which uses a noise-resilient voting algorithm for the selection. Henceforth, we refer to the input selection unit as the *voter unit*.

The interesting application scenarios are when the system is running continuously, where in every cycle of operation, each input unit produces a measurement. In many real scenarios, the incremental change in the physical quantity being measured (such as distance to an object, angular velocity, atmospheric pressure, etc. measured in an avionics control unit) from one cycle to the next is small and smooth. In each cycle, the voter unit has to select one of the input units as a *prime unit*. The selection should ensure a reliable output value by discarding the measurements having transient errors. In addition, permanently faulty units should be identified and isolated, so that they do not corrupt the feed to the controller. To do this effectively, the input selection of each cycle has to also take into account some information from the past cycles. If the measurement domain is $\mathcal{D}$, the number of input units is N and the information based on the past t cycles is used for the calculations, then the voting unit essentially calculates a function $f : \mathcal{D}^{tN} \mapsto \mathcal{D}$ in every cycle. In a more general situation, some self-reported parameters from the input units are also required for the decision making. It is also desired that, along with the selected measurement value, some reliability parameters of the output are also supplied to the controller. The reader may refer to Figure 1 which depicts an illustration of such a system.

One aspect of transient resilience of the system is to ensure that measurements with transient errors are discarded, without getting fed to the controller. On the other hand, measurements are usually taken from redundant units that are physically separated and there could be minor deviations in the measurements made by them. Hence, it is desirable to avoid *frequent switching* of the prime unit due to transient failures, to maintain better stability. Although these two aspects of transient resilience may seem conflicting, the assumption that the incremental variations in the measured quantity between consecutive cycles is small, makes it possible to strike a balance. If the prime unit selected for the previous cycle has a transient error in the current cycle, the algorithm can retain the output value of the previous cycle without switching the prime unit, as long as the age of the measurement is within the allowed limits. If an input unit is showing failure for many consecutive cycles beyond a *persistence limit*, then it has to be permanently isolated. These requirements make the design of an input selection unit for a controller different from a typical voting algorithm.

As explained by Butler [3] and Miller et al. [22], a major effort in the formal verification of a safety critical system goes into unambiguously capturing the requirements of the system using formal logic. A major contribution of this work is the formulation of a consistent set of requirements relevant to the application context presented above. We demonstrate the theoretical limits of achievable soundness and completeness. Our voter unit algorithm is proven to match the theoretical limits.

If the number of input units without a permanent fault falls below a certain threshold, then it is well known that fault identification cannot achieve completeness [7]. However, by fine-tuning the fault identification algorithm, in certain situations it is possible to identify with certainty, a subset of units as non-faulty and another subset of units as faulty. In such cases as well, we derive the theoretical limits of fault identification and our algorithm matches these limits. Such a finer fault identification will help in prolonging the duration of reliable operation of the voter unit, by avoiding mis-classification of a healthy input unit as permanently faulty.

When the algorithm is designed to prolong the duration of operation of the voter unit this way, there could be certain boundary cases wherein the prime unit of the previous cycle got isolated due to a permanent fault and it is not theoretically possible to identify with certainty even one unit as non-faulty to assign as the new prime unit. In such situations, if the number of units without a permanent fault is above *the minimum threshold for continuing the operation* of the voter unit, then for a short while our algorithm will change the state of the voter unit as un_id (unidentifiable), in anticipation of a recovery from transient errors, while continuing to provide the output from the previous cycle. The algorithm will take care of bounding the age of the output in such scenarios as well.

To make the system description and its requirements more precise, it is necessary to formally specify the assumed fault model and fault hypotheses [32] indicating the type, number and rate of faults that the system can tolerate while remaining operational [20, 27]. This is done in Section 2. In Section 3, we formulate the functional requirements of the system. In Section 4, we give an overall design of the voter unit. The voter unit is designed as a state transition system with three major functions: fault identification, handling of transient and permanent failures using a multi cycle history of faults, and generation of a reliable output, which will be fed to the controller.

In formal verification approaches like model checking, verification of such systems with large state space is challenging, since the proof of correctness is auto-generated by the model checker from the abstract system specification and the property specification. Hence, the verification time suffers from the state space explosion problem [5, Section 1.3.1]. In contrast,

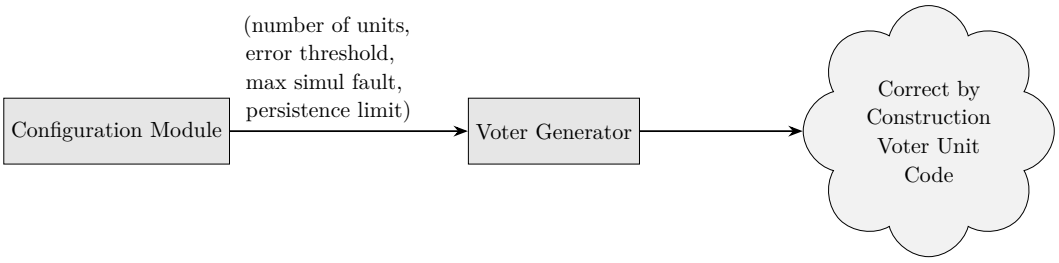


Figure 2 To adjust the reliability levels, some parameters are kept as designer configurable, with a permissible range of values for each. A voter unit code with the specified set of parameter values can be generated.

when machine-assisted theorem provers are used, the proof of satisfaction of the specification is created by the developer herself, interactively with guidance from the tool. While more developer time and effort are taken for the development of the proof, the verification time is only to machine check the proof created by the developer and confirm that it is correct [5, Chapter 1, Page 5]. Hence, the verification by the machine is fast.

This work is done using the Rocq interactive theorem prover [36]. Rocq theorem prover is based on the calculus of inductive construction and satisfies the de Bruijn criterion [4]. In addition, Rocq supports higher-order logics and dependent types, which makes it easy to express complex specifications succinctly. We have attempted to follow Rocq's design principle of defining functions carrying proof of its properties embedded with it, rather than completing a code first and then proving the properties. Some of the requirements are shown to be satisfied as invariants of the states, while the remaining requirements, which depend on the multi cycle behaviour of the system, are captured as invariants maintained during every state transition. To allow the designer to fine-tune the reliability parameters to match the application specific requirements, we maintain the definition of some basic reliability parameters in a separate configuration module, and the system implementation is parameterized by them. The code of the voter unit is synthesized with the selected values of these parameters (see Figure 2). In this approach, there is no gap between the implementation and the specification, since the executable code is directly extracted from a proven correct implementation. This contrasts with other formal verification approaches which give a formally verified model that needs to be then converted to an executable implementation.

2 System Description and Fault Model

In this section, we formally define the input and output formats of the voter unit and the fault model and fault hypothesis.

2.1 Input Format of the Voter Unit

In each cycle of operation, the voter receives input from N redundant input units that work synchronously and measure the same physical quantity that continuously varies over time. The value of N can be decided by the user. The voter unit maintains a unique identification number `uid` for each input unit based on the port of the voter unit interface to which the input unit is connected. From each input unit, the voter receives a `reading`, which is a 2-tuple consisting of a) the measured value `val`, and b) the self-identified health status of the unit `hw_hlth`. This kind of data format where a measurement is available with a health status is commonly used by sensors and measurement units used for critical applications [9, 13].

The measurement from a properly functioning input unit may be noisy, but within an acceptable threshold δ from the ground truth. An input unit can malfunction due to a transient or permanent fault.

2.2 Fault Model

As described in the input format, in each cycle, each input unit has a self-identified health status: `good` or `bad`. Another type of fault that is not self-identifying is a deviation greater than a fixed noise threshold δ from the ground truth of the physical quantity being measured. We call it a *deviation fault*. A faulty behaviour of an input unit is defined as an occurrence of any one of these two faults. A faulty behaviour is considered *transient* if it persists less than a consecutive number of cycles of operation, denoted by `persistence_lmt`. Otherwise, it is considered a *permanent fault*. For transient fault resilience, `persistence_lmt` should be greater than 1. It is assumed that the faulty behaviour is limited to the input units alone, and the hardware interface of the voter unit receiving readings from the input units is assumed to be non-faulty. Byzantine faults [20] are not considered in our formulation. We need to make a *simultaneous fault hypothesis* about the maximum number of simultaneous new faults that are anticipated in a cycle of operation. This assumption is stated below.

▶ **Hypothesis 1** (Simultaneous Fault Hypothesis). *In any cycle, among the input units that do not have a permanent fault before the cycle begins, at most* `max_simul_fault` *units can show faulty behaviour.*

The correctness of the voter unit's behaviour can be guaranteed only if a certain minimum number of input units without a permanent fault are present. Formally, we define a parameter `min_required` and say that the system satisfies *maximum permanent fault assumption* when there are at least `min_required` number of input units without a permanent fault. By *simultaneous fault hypothesis*, it is evident that `min_required` should be at least `max_simul_fault` + 1. The values of parameters δ, `persistence_lmt` and `max_simul_fault` can be decided by the user.

2.3 Output Format of the Voter Unit

The format of the output produced by the voter unit is a 4-tuple consisting of `voter_output`, `output_age`, `validity_status` and `presrvd_data`. The first three are for feeding to the controller, while `presrvd_data` is a feedback to the voter unit itself for decision making during state transitions.

In each cycle, the voter should select one of the input units as the *prime unit*. The `voter_output` consists of the `uid` of the prime unit and a `reading` from the prime unit. The `voter_output` is a current cycle measurement of the prime unit if the prime unit has no identified fault in the current cycle. Otherwise, the previous cycle `voter_output` is retained. The `output_age` is used to indicate the number of cycles since when the voter output measurement has not been updated. The `validity_status` is an indication of the reliability of the `voter_output` and can be `valid`, `un_id` (unidentifiable) or `not_valid`. The `presrvd_data` consists of the `uid`, `reading` and accumulated fault history of the prime input corresponding to the `voter_output`.

3 Formulation of Functional Requirements

The functional requirements of the generic data selection and voting unit is described in this section. The requirements are broadly classified into three categories: (i) related to fault identification, (ii) related to isolation of permanently faulty units and (iii) related

to output generation. We provide an analysis of the theoretical limits achievable for fault identification and show that the requirements formulated are tight with respect to these limits. The proof of correctness of fault identification rules presented in a conventional way in this section are machine verified in our Rocq implementation. The consistency of the set of requirements is verified in Rocq, and the code of the voter unit produced is proven to be correct by construction satisfying all the requirements.

3.1 Fault Identification

In each cycle, the voter unit should analyze the input received from the input units and record the information of faults identified by the fault identification algorithm. As a method of recording the faults, the voter unit will maintain a `risky_count` corresponding to each input unit, which indicates the number of consecutive cycles of operation (including the current cycle) in which the measurement from the unit is not identified as non-faulty. Before explaining the details of requirements related to fault identification, we need to first give the definitions of soundness and completeness of fault identification.

3.1.1 Soundness and completeness

As the actual value (ground truth) of the physical quantity being measured is unknown, units with deviation faults have to be identified using mutual deviation checks. As per the fault model, a non-faulty measurement value can have a deviation of δ from the ground truth, in a positive or negative direction. Hence, a deviation of 2δ between two inputs is allowed and cannot be identified as a fault.

▶ **Definition 1** (miscomparison). *A deviation of more than 2δ between measurements in a cycle from a pair of input units is a miscomparison.*

A miscomparison indicates that at least one of the two measurements involved in the comparison is faulty. Since it is theoretically impossible to identify all deviation faults in certain scenarios [7], we want to classify the units to have their `miscomparison_status` to be one among `miscomparing`, `not_miscomparing` and `maybe_miscomparing`.

We start by analysing a scenario of fault identification that will justify the definitions of soundness and completeness of fault identification that follow.

▷ Claim 2. There are situations where the measurement of an input unit u deviates 3δ from the ground truth and it is still impossible to detect the unit as faulty.

Proof. Suppose that unit u gives a measurement $x+3\delta$, while other units give a measurement value $x+\delta$. Now, consider two scenarios (i) the (unknown) ground truth is x and (ii) the ground truth is $x+2\delta$.

In both scenarios, the mutual deviation check will not show any miscomparison. In the first case, the measurement of u deviates 3δ from the ground truth and in the second case, its deviation is within the noise threshold δ from the ground truth. Since the ground truth is unknown, it is impossible to distinguish between these two scenarios. ◁

The above claim justifies the definitions of soundness and completeness of the fault identification stated below. We will discuss the acheivability of these in Section 3.1.2.

▶ **Definition 3** (Soundness and Completeness of Deviation Fault Identification).

- *(Soundness a) If the* `miscomparison_status` *of a unit is* `miscomparing`*, its measurement deviates more than δ from the (unknown) ground truth.*

- *(Soundness b) If the* `miscomparison_status` *of a unit is* `not_miscomparing`*, then the deviation of the measurement of that unit from the ground truth is at most* 3δ.
- *(Completeness a) If the measurement of a unit deviates more than* 3δ *from the ground truth, then its* `miscomparison_status` *should be* `miscomparing`.
- *(Completeness b) If the deviation of the measurement of a unit from the ground truth is at most* δ*, its* `miscomparison_status` *should be* `not_miscomparing`.

3.1.2 Requirements of soundness and completeness of fault identification

The requirement regarding the handling of self-identifying faults is straightforward. Input units with self-identifying `bad` health must be identified as faulty. The following basic correctness requirement for fault identification algorithm is the following.

R1. For an input unit which is not yet marked as permanently faulty,

- `risky_count` is either zero or one more than its `risky_count` in the previous cycle and
- `risky_count` is incremented in a cycle if and only if its self identifying health status is `bad` or its `miscomparison_status` is `miscomparing` or `maybe_miscomparing`.

Now, we will identify some conditions under which the completeness property of fault identification cannot be achieved, so that the requirements on the completeness can be fine tuned. One such scenario is when among the input units that do not have a permanent fault, there are not enough units without self-identifying faults. We will make this statement more precise shortly. Among input units without a permanent fault, if there are k units with self-identifying `bad` health in a cycle, then by *simultaneous fault hypothesis*, there can be at most $\texttt{max_simul_fault} - k$ units with non-self-identifying deviation faults. Let $\texttt{mis_flt_lmt} = \texttt{max_simul_fault} - k$.

▷ Claim 4. There are situations in which the number of input units that do not have a permanent fault and having good health is $2 * \texttt{mis_flt_lmt}$ and yet it is not possible to achieve completeness of deviation fault identification.

Proof. Consider a situation where `mis_flt_lmt` is 2 and there are 4 units u_1, u_2, u_3, u_4 without a permanent fault yet and with `good` health. Suppose u_1 and u_2 have the same measurement x and u_3 and u_4 have the same measurement y, which is more than 3δ away from x. There are two possible scenarios: (i) both u_1 and u_2 are faulty, while y is within δ of the ground truth (ii) both u_3 and u_4 are faulty, while x is within δ of the ground truth.

Here, since $|x - y| > 3\delta$, it is clear that either x or y deviates more than δ from the ground truth. However, since the ground truth is unknown, it is impossible to distinguish which among these actually occurred. Hence, even when the number of input units that do not have a permanent fault and having good health is $2 * \texttt{mis_flt_lmt}$, completeness cannot be achieved. ◁

▶ **Definition 5** (minimum surviving units assumption)**.** *The system is said to satisfy the minimum surviving units assumption when among input units that do not have a permanent fault in a cycle, at least* $2 * \texttt{mis_flt_lmt} + 1$ *units have* `good` *health status.*

By Claim 4, the following requirement regarding completeness of fault identification is tight with respect to the theoretical limit.

R2. When the system satisfies the minimum surviving units assumption, the `miscomparison_status` of each unit should be either `miscomparing` or `not_miscomparing` and this classification should satisfy the completeness requirements given in Definition 3.

It can be shown that this requirement is always satisfiable. We will address this point later. The most important requirement of our deviation fault identification algorithm is the following.

R3. The soundness conditions specified in Definition 3 must always be satisfied by the fault identification algorithm.

3.1.3 Avoiding mis-classifications to extend operational life

If the minimum surviving units assumption is not satisfied, our main objective is to satisfy the soundness requirements given in Definition 3. For this, it suffices to trivially declare all units as `maybe_miscomparing` when the *minimum surviving units assumption* is not satisfied. However, this may lead to an early declaration of the input units as permanently faulty and may cause the system to fail to satisfy the *maximum permanent fault assumption*. Hence, let us explore some more scenarios in which partial fault identification can be achieved.

▷ Claim 6. If the measurement value of an input unit u miscompares with at least `mis_flt_lmt` $+ 1$ other input units without a permanent fault yet and with `good` health, then the measurement of u deviates more than δ from the ground truth. Further, there are situations, where a unit is miscomparing with `mis_flt_lmt` other units and still the unit cannot be identified as faulty.

Proof. Among the `mis_flt_lmt` $+ 1$ other units with which unit u miscompares, by the *simultaneous fault hypothesis*, there must be at least one non-faulty unit u'. Since measurement of u' is within δ from ground truth and the measurements of u and u' deviate more than 2δ, it follows that the measurement of u definitely deviates more than δ from the ground truth. Hence, u is faulty. To see the second part of the claim, it suffices to consider the scenario described in the proof of Claim 4. ◁

Now, let us consider another interesting scenario where a partial fault identification is possible. To simplify our presentation, we make the following definitions.

▶ **Definition 7** (`miscomparing_list`, `rem_mis_flt_lmt`)**.** `miscomparing_list` *in a cycle is the list of input units without a permanent fault yet and with* `good` *health and whose measurement miscompares with at least* `mis_flt_lmt` $+ 1$ *other such units. We define* `rem_mis_flt_lmt` $=$ `mis_flt_lmt` $-$ |`miscomparing_list`|.

▶ **Definition 8** (identified fault)**.** *A unit is said to have an identified fault in a cycle if its health status is* `bad` *or it is in the* `miscomparing_list`.

▷ Claim 9. Let u be a not yet permanently faulty input unit without an identified fault. If the deviation of the measurement of u is within 2δ from `rem_mis_flt_lmt` other such units, then the measurement of u is within 3δ of the ground truth. Further, there are situations where the measurement of such a unit u is within 2δ from exactly `rem_mis_flt_lmt` -1 other units and still it is impossible to assign a miscomparison status `miscomparing` or `not_miscomparing` to u satisfying the soundness requirements.

Proof. By simultaneous fault hypothesis, `rem_mis_flt_lmt` is an upper bound on the remaining number of units with faults among units with health status `good`, after applying the rule in Claim 6. Hence, if the deviation of the measurement of a unit u which does not have an *identified* fault is within 2δ from `rem_mis_flt_lmt` other such units, then we can conclude that either u or at least one among them is non-faulty. In both these cases, we can conclude that the unit u has a measurement within 3δ from the ground truth.

To see the second part of this claim, it is sufficient to analyze a similar situation as given in the proof of Claim 4. Suppose `mis_flt_lmt` is 2 and there are 4 units u_1, u_2, u_3, u_4 without a permanent fault yet and with `good` health. Suppose u_1 and u_2 have the same measurement x and u_3 and u_4 have the same measurement y, which is more than 3δ away from x. Consider two possible scenarios: (i) the ground truth is x (ii) the ground truth is y. Here, observe that `rem_mis_flt_lmt` is the same as `mis_flt_lmt`, which is 2. To satisfy the soundness condition, in the first scenario, miscomparison status of neither u_1 nor u_2 can be assigned as `miscomparing`. In the second scenario, their miscomparison status cannot be assigned as `not_miscomparing`. The situation of u_3 and u_4 is symmetric.

Since it is impossible to distinguish between the two scenarios without knowing the ground truth, the miscomparison status of none of the units can be set to `miscomparing` or `not_miscomparing` ensuring the soundness requirements. ◁

Claim 6 and Claim 9 show that the two requirements stated below are tight with respect to the theoretical limits.

R4. If the measurement value of an input unit miscompares with at least `mis_flt_lmt` + 1 other input units without a permanent fault yet and with `good` health, then it should have `miscomparison_status` as `miscomparing`.

R5. Among the not permanently faulty input units which do not have an identified fault, if the measurement of a unit is within 2δ from at least `rem_mis_flt_lmt` others, then its `miscomparison_status` should be `not_miscomparing`.

Now, the following is an interesting observation.

▶ **Proposition 10.** *When the system satisfies the minimum surviving units assumption, requirements R4 and R5 are sufficient to guarantee requirements R2 and R3.*

Proof. Suppose that the system satisfies the minimum surviving units assumption. Then $2 * \texttt{mis_flt_lmt} + 1$ input units are available among units without a permanent fault yet and with `good` health. Consider such an arbitrary unit u. Among the other $2 * \texttt{mis_flt_lmt}$ units other than u, if there are at least $\texttt{mis_flt_lmt} + 1$ units with which u miscompares, then by Claim 6, measurement of u deviates more than δ from the ground truth. Otherwise, there are at most `mis_flt_lmt` other units with which u miscompares and hence there are at least $\texttt{mis_flt_lmt} \geq \texttt{rem_mis_flt}$ units whose measurement is within 2δ from that of u. In this case, by Claim 9, the measurement of u is within 3δ deviation from the ground truth. In the first case, by R4 the miscomparison status of u will be `miscomparing` and the second case, by R5 u will have miscomparison status `not_miscomparing`. In both cases, the soundness and completeness properties are satisfied. Since this can be done for an arbitrary u, R2 will be satisfied. Since soundness is ensured, R3 also holds. ◀

▶ Note 11. Requirements R2 and R3 together means that under minimum surviving units assumption, all units are assigned miscomparison status `miscomparing` or `not_miscomparing` and the classification is sound and complete. By Proposition 10, this can be ensured by satisfying requirements R4 and R5. Moreover, we can algorithmically ensure R4 and R5 by applying the straightforward conditions in Claim 6 and Claim 9.

3.2 Isolation of Permanently Faulty Units

As discussed in the previous section, in each cycle the voter unit uses its fault identification algorithm to identify the fault status of each input unit. The voter unit maintains some information to keep track of the accumulated fault status of each unit. This information will be used to distinguish between transient faults and permanent faults in order to avoid corruption of the feed to the controller in future from a permanently faulty unit.

One such information maintained is the `risky_count` of an input unit, which needs to be updated as mentioned in requirement R1. An input unit with a non-zero `risky_count` which is less than `persistence_lmt` is regarded to have a *transient fault*. If the transient fault clears before it reaches `persistence_lmt`, then the `risky_count` is reset to zero. A sequence of identified faulty behaviour in consecutive cycles, leading to the `risky_count` becoming equal to `persistence_lmt` is regarded as a *permanent fault*.

For each input unit, the voter unit also maintains an `isolation status`. The following are the requirements related to the isolation of a unit.

R6. A unit is isolated if and only if the `risky_count` of the unit reaches the predefined threshold indicated by `persistence_lmt`.

R7. If an input unit is isolated in a cycle of operation, it continues to remain isolated for all future cycles.

3.3 Output Generation Related Requirements

Using the current cycle input data from each input unit and based on the updated status information of each input unit, the voter generates an output, which is used as feed to the controller. As mentioned in the output format, the feed to the controller consists of `voter_output`, `output_age` and `validity_status`.

3.3.1 Voter output and prime unit switching

A `unit_output` consists of the `uid` of a unit and a `reading` received from it. To explain the requirements regarding the output, the following definition will be useful.

▶ **Definition 12** (healthy data). *A* `healthy_data` *is a* `unit_output` *with* `good` *health such that, in the cycle of its measurement the input unit with the corresponding* `uid` *is not isolated and its* `miscomparison_status` *is* `not_miscomparing`*.*

The `voter_output` in a cycle is the selected `unit_output` and the `uid` present in it is the prime unit of the cycle. In case of a transient fault in the prime unit, the `voter_output` in the previous cycle will remain the `voter_output` for current cycle, until the fault disappears or becomes permanent. One basic requirement is that if a fault is identified, it should be contained within the faulty unit and should not be allowed to corrupt the input to the controller. This requirement is stated below.

R8. The voter always outputs a `healthy_data`.

If the prime unit of the previous cycle gets isolated in the current cycle and another healthy unit is available, then the voter should select another input unit to provide the feed for the controller. However, as mentioned in the introduction, a requirement in this context is that frequent change of the prime unit used for output generation must be avoided. This requirement is formulated below.

R9. Switching of the prime unit occurs in a cycle only if the prime unit of the previous cycle got isolated.

The following proposition shows that if the requirements defined so far are satisfied, switching of prime unit will not happen frequently.

▶ **Proposition 13.** *By satisfying the requirements R1, R6, R8 and R9, it is ensured that the input unit selected as the prime unit shall not change for at least* `persistence_lmt` *number of cycles.*

Proof. Consider the time when an input unit u is just selected as the prime unit. By R8, u has health status `good` and miscomparison status `not_miscomparing`. From R9, a new prime unit is selected only when u gets isolated. However, by requirements R1 and R6, it must take at least `persistence_lmt` cycles for this to happen. ◀

3.3.2 Validity status related requirements

The correctness of the voter behaviour (fault identification, isolation and output generation) is guaranteed only if the *maximum permanent fault assumption* is satisfied. The voter maintains a `validity_status` to indicate the reliability of the output generated. The validity status can be `valid`, `un_id`(unidentifiable) or `not_valid`.

The validity status `not_valid` indicates that the voter output is not reliable because the *maximum permanent fault assumption* does not hold and hence, no meaningful fault identification is possible. As mentioned in the introduction and in section 3.1.3, our voter unit design does a finer fault identification, even if it is incomplete, to delay the situation of the system failing to satisfy the *maximum permanent fault assumption*. As a consequence, there could be situations wherein the prime unit of the previous cycle is isolated in the current cycle and there is no input unit with `healthy_data` to be used as the new prime unit, satisfying R8. This condition occurs when the fault identification logic is failing to identify even one input unit with `healthy_data` because the condition mentioned in R5 is not satisfied. In such scenarios, if *maximum permanent fault assumption* still holds, there is a possibility that the faulty condition is transient and the voter may be able to recover from this condition in a few cycles. Hence, the voter unit can choose to retain the `voter_output` of the previous cycle. The `un_id` validity status is used to capture this situation. The validity status `valid` represents a higher level of reliability than `un_id`, as we will explain shortly. The requirements regarding validity status are captured below.

R10. The validity status of the voter is `not_valid` if and only if the system does not satisfy the *maximum permanent fault assumption*.

R11. The validity status of the voter is `un_id` if and only if the *maximum permanent fault assumption* is satisfied and the prime unit of the previous cycle is isolated and no input unit is identified to be providing a `healthy_data`.

R12. If the system satisfies the *maximum permanent fault assumption* and the *minimum surviving units assumption* (see Definition 5), then the validity status of the voter must be `valid`.

R13. If the *maximum permanent fault assumption* is satisfied, and if at least one unit is identified to be providing a `healthy_data`, then the validity status of the voter must be `valid`.

3.3.3 Output age related requirements

The `output_age` is used by the voter to indicate the number of cycles since when the voter output measurement has not been updated. A basic correctness requirement related to `output_age` updation is the following:

R14. Unless the `validity_status` is `not_valid`, `output_age` is either zero or one more than the previous cycle `output_age`. Further, if validity status is `valid`, then the `output_age` is equal to the `risky_count` of the prime unit.

An upper bound on `output_age` is required to ensure that the feed to the controller is a measurement with a bounded lag. When the validity status is `valid`, our requirements for the `output_age` are as follows.

R15. The `output_age` is zero if and only if the validity status is `valid` and voter output is a current cycle measurement.

R16. If the validity status is `valid`, then $\texttt{output_age} \leq \texttt{persistence_lmt} - 1$.

It is interesting to note that from the requirements stated so far, an upper bound of $2 * (\texttt{persistence_lmt} - 1)$ is guaranteed for `output_age` when the validity status is `un_id`. To prove this, we will use a relation that exists between the `risky_count` of non isolated input units and the `output_age` when the voter validity is other than `not_valid`. The relation is stated below.

▷ Claim 14. Suppose all the requirements are satisfied. Then, unless the validity status is `not_valid`, for each input unit u which is not isolated, $\texttt{output_age} - \texttt{risky_count}$ of $u <$ `persistence_lmt`.

Proof. We will show that this invariant is satisfied in each cycle if all the requirements (specifically, R1, R6, R10, R11, R14, and R16) are satisfied. By requirements R6 and R10, if the validity status becomes `not_valid`, it must remain so in the future. Initially, all units have `risky_count` zero and the `output_age` is also zero. Since $\texttt{persistence_lmt} > 0$, the relation holds initially for all input units. Now, consider a cycle in which the validity status is different from `not_valid` after the cycle's fault identification. If the validity status is `valid`, then by requirement R16, $\texttt{output_age} < \texttt{persistence_lmt}$ and hence the claim holds. Otherwise, the validity status must be `un_id`. In this case, consider an arbitrary input unit u which remains non-isolated after the cycle. It suffices to show that if the relation holds for u before the cycle, it holds after the cycle as well.

By R11, u cannot be providing a `healthy_data` in the current cycle. Therefore, by requirement R1, the `risky_count` of u must be exactly one more than its value in the previous cycle. Moreover, by R14, `output_age` can increase by at most one. Hence, as required, if the relation holds for u before the cycle, it holds after the cycle as well. ◁

The following proposition is now easy to prove.

▶ **Proposition 15.** *If the system satisfies requirements R1 to R16, then the* `output_age` *is at most* $2 * (\texttt{persistence_lmt} - 1)$ *as long as the maximum permanent fault assumption holds.*

Proof. By requirement R10, unless the validity status is `not_valid`, the maximum permanent fault hypothesis holds and so, there are at least $\texttt{min_required} > 0$ non-isolated units. Consider one such non-isolated unit u. By Claim 14, $\texttt{output_age} - \texttt{risky_count}$ of $u \leq \texttt{persistence_lmt} - 1$. Further, by R6, the `risky_count` of u is at most $\texttt{persistence_lmt} - 1$. Hence, $\texttt{output_age} < 2 * (\texttt{persistence_lmt} - 1)$. ◀

▶ Note 16. It is not obvious that the requirements R1 to R16 are consistent. The consistency is established by machine assisted proof created using Rocq. En route, we also obtained machine generated proofs of the first part of Claim 6, the first part of Claim 9, Proposition 10, Proposition 13 and Claim 14.

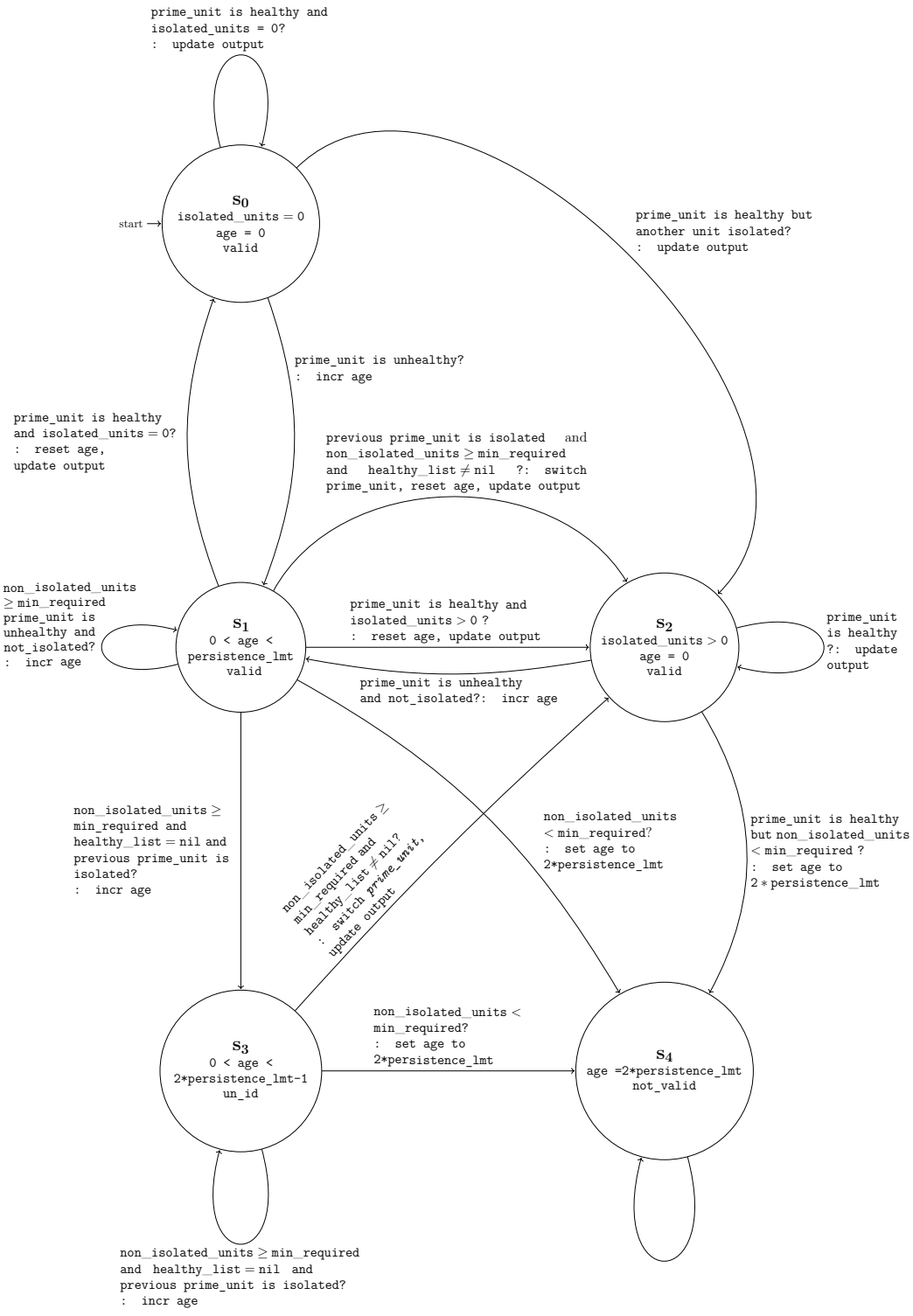


Figure 3 State transition diagram of the voter unit. To avoid cluttering, the update in the isolation status of units and the number of isolated units during the transitions are not shown. The update in validity status is not explicitly shown during the transitions, but can be understood from the states.

4 Design of the Voter Unit

As described in the voter output format, the voter unit provides a `voter_output` for the controller along with an indication of the `validity_status` and `output_age`. The design of the voter unit has three key aspects: fault identification, handling of transient and permanent faults and computation of the voter output. The voter unit is designed as a state transition system. The configurable parameters of the voter unit are: `num_units` to indicate the count of input units, δ to indicate the noise threshold, `persistence_lmt` used for isolation of permanent faulty units and `max_simul_fault` to indicate the maximum number of simultaneous faults that are anticipated in a cycle of operation.

In every transition, the voter unit will update the fault status (`unit_status`) of each input unit by performing fault identification on the current cycle reading. The `unit_status` includes the `isolation_status`, `miscomparison_status` and the `risky_count` of the corresponding input unit. The logic used in the fault identification algorithm is an exact translation of the conditions specified in requirements R4 and R5. If both conditions do not apply, then the unit will be classified as `maybe_miscomparing`. This method will satisfy requirements R2 - R5, as explained in Section 3.1.2 and 3.1.3. After the fault identification, the update in `risky_count` is done as described in R1. Using this, the isolation of permanently faulty units is done as mentioned in requirements R6 and R7.

We explain the voter unit behaviour using five abstract states, as shown in the state transition diagram in Figure 3. The description of the states are as follows.

- S_0: This is the initial state where there are no isolated inputs, the `output_age` is zero and the `validity_status` is `valid`. The output value will be the current cycle measurement of the prime unit.
- S_1: This state occurs when the prime unit used for `voter_output` generation is having a transient fault. The previous cycle `voter_output` is retained to satisfy R8, and `output_age` is incremented on transition to this state.
- S_2: This state occurs when the number of isolated units is non-zero, but non-isolated units are at least `min_required`. There is at least one unit providing `healthy_data` and one such unit is selected as the prime unit. Further, the `output_age` is zero, indicating that the output value is a current cycle measurement from prime unit, and `validity_status` is `valid`.
- S_3: This state occurs when the input unit used for `voter_output` generation in the previous cycle got isolated and there are no input units providing `healthy_data`, even though the number of non-isolated units is at least `min_required`. This state satisfies the premise of the requirement R11 and hence `validity_status` is `un_id`. The previous cycle `voter_output` is retained to satisfy R8, and `output_age` is incremented on transition to this state.
- S_4: This state occurs only when the number of non-isolated units is below `min_required`. In this state, the validity status is `not_valid` as per requirement R10.

▶ Note 17. It can be verified from the description of states that the requirements R6, R8, R10, R11, R13, R15, R16 and the second part of requirement R14 are invariants for all states.

The state transitions are summarized below.

- From the initial state S_0, there are three transitions. These are a) to state S_1, when the prime unit used for `voter_output` generation is having a transient fault and b) to state S_2, when the prime unit is healthy and one of the other units is isolated. The system remains in state S_0 as long as the initially selected prime unit is healthy and none of the other units are isolated.

- From states S_1, S_2 and S_3, a transition to S_4 happens if and only if the *maximum permanent fault assumption* is not satisfied due to isolation of input units.
- From state S_1, there are five more transitions. These transitions are a) to state S_1 when the prime unit has a transient fault b) to state S_2 when the prime unit of the previous cycle is isolated and there is at least one healthy unit to be identified as the prime unit c) to state S_2 when the prime unit of the previous cycle recovers from a transient fault and at least one input unit is isolated d) to state S_0 when prime unit of previous cycle recovers from a transient fault and none of the input units are isolated e) to state S_3 when prime unit of previous cycle is isolated and there are no healthy units to be identified as a prime unit.
- From state S_2, there is a transition to the same state when the prime unit is healthy. There is also a transition to state S_1 when the selected prime unit has a transient fault.
- From state S_3, there are two more transitions. These are a) to state S_3 when there are no healthy units and b) to state S_2 when one of the input units recovers from a transient fault and becomes the prime unit.
- Once the system reaches the state S_4, it should remain there, in order to satisfy the requirements R7 and R10.

▶ Note 18. It can be seen that the requirements R1 - R5, R7, R9, R12, and the first part of requirement R14 are invariants of the state transition rules.

5 Implementation in Rocq

We have used the design given in Section 4 for our Rocq implementation. We have used Rocq's support for higher-order logics and dependent types extensively while formally expressing the requirements mentioned in Section 3. Our implementation follows a correct-by-construction approach by defining the voter state and the state transition rules in such a way that they carry with them the proof terms of the invariants they satisfy. This makes sure that when these definitions are completed and are verified by Rocq, the invariants will certainly hold by the implementation. Since Rocq is based on the Calculus of Construction, all proofs are fully constructable, rather than being existential. The total time taken for verification in Rocq for properties R1 to R16 is approximately 23 seconds on a laptop with Debian 12 OS, 32GB RAM, 64-Bit Processor (Intel® Core™ Ultra 5 125U × 14).

The verified code obtained in Rocq is extractable to OCaml/Haskell/Scheme using the provisions in Rocq [35]. The number of lines of the extracted OCaml code is 362. The execution time of the extracted OCaml code for a single cycle state transition is around 3 milliseconds.

To see a detailed description of our implementation using Rocq, the reader may refer to the appendix of the extended arXiv version [1]. However, for quick reference the definition of the voter state has been included in the appendix of this paper. It may be noticed that the definition of `voter_state` carries embedded proof terms of properties corresponding to requirements R8, R10, R11, R13, R15, R16 and the second part of requirement R14. The requirement R6 is an invariant of `unit_status`, which is a field inside the `voter_state`. The state transition function is named as `voter_state_transition`. It carries proof terms of properties corresponding to requirements R1 - R5, R7, R9, R12, and the first part of requirement R14. The entire source code of our implementation is available in GitHub[1].

[1] `https://github.com/arifali-ap/Fault_tolerant_voter.git`

6 Discussion and Future Directions

Our work gives the design and synthesized code of a voter unit that feeds a control system. The synthesized code is correct by construction. Brown et al. [6] and Krishnan et al. [17] report verification of voting logic used in safety-critical applications using other formal verification paradigms, for systems with at most four redundant units. The properties they verified are limited to certain state transition properties expressed using LTL and CTL for their abstract transition system. In our implementation, the support available in Rocq for higher order logics and dependent types helped us in succinctly specifying a more extensive set of requirements.

Brown et al. [6] report scalability issues and a verification time of 14000 seconds for some properties. Our design is parameterized by the number of units and some reliability parameters. The values of the parameters can be appropriately fixed by the user during the extraction of the code from Rocq to OCaml/Haskell. Since we use the theorem prover based approach, our verification time does not increase with increase in number of units. The total time taken for verification of properties is 23 seconds only. Further, while Brown et al. [6] and Krishnan et al. [17] produce verified models, our work gives a verified implementation.

In fault tolerant systems with a multi-level hierarchical design, redundancy management is required in each level of hierarchy. An extension in this direction is interesting. Further, one may have to deal with fault handling of asynchronous redundant units in such systems. For example, in a distributed fault-tolerant system, the control unit may receive feed from multiple asynchronous voting units, each working with independent logical clocks with small deviations. Each of these voting units may be receiving data from redundant input units. Synthesis of verified code for identifying and handling faults in such asynchronous systems is another direction for future work.

References

1 Arif AP, Jasine Babu, and Deepa Sara John. A Correct by Construction Fault Tolerant Voter for Input Selection of a Control System, 2025. URL: https://arxiv.org/abs/2509.22236.

2 A. Aviziens. Fault-tolerant systems. *IEEE Transactions on Computers*, C-25(12):1304–1312, 1976. doi:10.1109/TC.1976.1674598.

3 Ricky Butler and Sally Johnson. *Formal Methods For Life-Critical Software*, pages 319–329. Aerospace Research Central, October 1993. doi:10.2514/6.1993-4516.

4 Adam Chlipala. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*. The MIT Press, 2013.

5 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

6 Samar Dajani-Brown, Darren Cofer, Gary Hartmann, and Steve Pratt. Formal modeling and analysis of an avionics triplex sensor voter. In *Model Checking Software*, pages 34–48, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-44829-2_3.

7 Daniel Davies and John F. Wakerly. Synchronization and matching in redundant systems. *IEEE Transactions on Computers*, C-27:531–539, 1978. doi:10.1109/TC.1978.1675144.

8 J Gloria Davis. An analysis of redundancy management algorithms for asynchronous fault tolerant control systems. Technical report, Ames Research Center, California, NASA, 1987.

9 Analog Devices. ADIS1675 Datasheet. Accessed: 2025-07-10. URL: https://www.analog.com/en/products/adis16575.html.

10 Ben L DiVito, Ricky W Butler, and James L Caldwell. Formal design and verification of a reliable computing platform for real-time control. Phase 1: Results. Technical report, NASA Langley Research Center, 1990.

11 Jack Goldberg, Milton W. Green, William H. Kautz Karl N. Levitt, P. Michael Melliar-Smith, Richard L. Schwartz, and Charles B. Weinstoc. Development and analysis of the software implemented fault-tolerance (sift) computer. Technical report, SRI International, Menlo Park, California, 1984.

12 E.F. Hitt and D. Mulcare. Fault-tolerant avionics. In C.R. Spitzer and Spitzer, editors, *Digital Avionics Handbook (1st ed.)*, chapter 28, pages 1–28. CRC Press, 2006. doi:10.1201/9781420036879.

13 Honeywell. GG1320 User manual. Accessed: 2025-07-10. URL: https://aerospace.honeywell.com/content/dam/aerobt/en/documents/learn/products/sensors/user-manuals/GG1320-usermanual.pdf.

14 Benjamin F. Jones and Lee Pike. Modular model-checking of a byzantine fault-tolerant protocol. In *NASA Formal Methods*, pages 163–177, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-57288-8_12.

15 M. A. Kassab, H. S. Taha, S. A. Shedied, and A. Maher. A novel voting algorithm for redundant aircraft sensors. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pages 3741–3746, 2014. doi:10.1109/WCICA.2014.7053339.

16 J.C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 547–550, 2002.

17 Ranjani Krishnan. Simulation and application of voting algorithms for safety critical systems. In *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1145–1150, 2020. doi:10.1109/ICECA49313.2020.9297585.

18 Ranjani Krishnan, Ashutosh Gupta, Nitin Chandrachoodan, and V R Lalithambika. Formal verification of clock synchronization algorithms for aerospace systems. In *2024 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1603–1608, 2024. doi:10.1109/ICECA63461.2024.10800832.

19 Ranjani Krishnan, Ashutosh Gupta, Nitin Chandrachoodan, and VR Lalithambika. Formal verification of voting algorithms for safety critical systems using two approaches. In *2024 IEEE Space, Aerospace and Defence Conference (SPACE)*, pages 211–215. IEEE, 2024.

20 Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982. doi:10.1145/357172.357176.

21 Patrick Lincoln and John Rushby. The formal verification of an algorithm for interactive consistency under a hybrid fault model. In *Computer Aided Verification*, pages 292–304, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. doi:10.1007/3-540-56922-7_24.

22 Steven P. Miller, Alan C. Tribble, and Mats P. E. Heimdahl. Proving the shalls. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, pages 75–93, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

23 Paul S Miner. Verification of fault-tolerant clock synchronization systems. Technical report, NASA Langley Research Center, 1993.

24 M. Nicolaidis. *Soft Errors In Modern Electronic Systems*. Springer New York, NY, January 2011. doi:10.1007/978-1-4419-6993-4.

25 S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995. doi:10.1109/32.345827.

26 L. Pike. A note on inconsistent axioms in rushby's "systematic formal verification for fault-tolerant time-triggered algorithms". *IEEE Transactions on Software Engineering*, 32(5):347–348, 2006. doi:10.1109/TSE.2006.41.

27 D. Powell. Failure mode assumptions and assumption coverage. In *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*, pages 386–395, 1992. doi:10.1109/FTCS.1992.243562.

28 Vincent Rahli, Ivana Vukotic, Marcus Völp, and Paulo Esteves-Verissimo. Velisarios: Byzantine fault-tolerant protocols powered by Coq. In *Programming Languages and Systems*, Cham, 2018. Springer International Publishing.

29 J. Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Transactions on Software Engineering*, 25(5):651–660, 1999. doi:10.1109/32.815324.

30 J.M. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, 1993. doi:10.1109/32.210304.

31 John Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 304–313, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/197917.198115.

32 John Rushby. An overview of formal verification for the time-triggered architecture. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 83–105, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45739-9_7.

33 John Rushby and Frieder VonHenke. Formal verification of a fault tolerant clock synchronization algorithm. Technical report, NASA Langley Research Center, 1989.

34 Indranil Saha, Suman Roy, and S. Ramesh. Formal verification of fault-tolerant startup algorithms for time-triggered architectures: A survey. *Proceedings of the IEEE*, 104:1–19, March 2016. doi:10.1109/JPROC.2016.2519247.

35 Rocq Core Team. Program extraction. Accessed: 2025-07-08. URL: https://rocq-prover.org/doc/master/refman/addendum/extraction.html.

36 Rocq Core Team. Rocq prover. Accessed: 2025-07-08. URL: https://rocq-prover.org/.

A Voter State Definition and its Invariants

A basic description of the `voter_state` data type and its invariants is provided here. For a more detailed explanation of the individual fields of the data type, and the details of the state transition function, the reader may refer to Appendix of the extended arXiv version [1].

Basic Data Types

Each input unit is assumed to have a `unit_id`. This is defined as follows.

```
Inductive unit_id := uid_con : nat -> unit_id.
```

The `signal_health` type given below is used to represent the self-declared health status of an input unit (`good` or `bad`).

```
Inductive signal_health := | bad  : signal_health
                           | good : signal_health.
```

The `signal` type given below is used to represent a measurement from an input unit.

```
Record signal := signal_build {
    val     : nat;
    hw_hlth : signal_health;  }.
```

A `reading` of type `signal` and `uid` of type `unit_id` together constitute a `unit_output` data type. This represents the information received by the voter from each input unit.

```
Record unit_output := unit_output_build {
    reading : signal;
    uid     : unit_id;                  }.
```

An accumulated status of the fault identification and isolation of each input unit is maintained by the voter using the data type below.

```
Record unit_status := unit_status_build                              {
    iso_status     : isolation;
    miscomp_status : miscomparison_status;
    risky_count    : nat;
    pf_risky_count : risky_count <= persistence_lmt /\
                     (risky_count = persistence_lmt <->
                                          iso_status = isolated);  }.
```

The proof term `pf_risky_count` in `unit_status` encodes requirement R6 as an invariant. The voter sees the data type `unit_data` from each input unit. This is a record of unit_output and unit_status as defined below.

```
Record unit_data := unit_data_build                                   {
    u_output   : unit_output;
    u_status   : unit_status;
    pf_healthy : (risky_count u_status) = 0
                 <-> ( hw_hlth (reading u_output) = good
                    /\(iso_status u_status)     = not_isolated
                    /\(miscomp_status u_status) = not_miscomparing ); }.
```

The proof term `pf_healthy` carries the invariant that the `risky_count` is zero if and only if the data of the input unit is a `healthy_data`.

Voter State

The voter receives input from a predefined number of input units, `num_units` $\in \mathbb{N}$. To indicate that the voter sees inputs from units with unique ids, we define a data type `uid_list` which is a list of `unit_id`, with an embedded proof that there is no duplication in the list.

```
Record uid_list := uid_list_build  {
      l    : list unit_id;
      pf_l : NoDup l;              }.
```

A list named `u_ids` of type `uid_list` is the concrete list of `unit_id` defined for the system in which the unit ids correspond to the sequence of numbers from 1 to `num_units`. The voter is implemented as a state transition system. States are represented using the `voter_state` data type which is defined below.

```
Record voter_state := voter_state_build                                {
    u_data_lst    : list unit_data;
    (* On unit switching, the next unit is selected as per the
       order in u_data_lst *)
    pf_ud_lst     : get_u_ids_of_snsr_data u_data_lst = u_ids.(l);
    voter_output  : unit_output;
    voter_validity: validity_status;
    output_age    : nat;
    presrvd_data  : unit_data; (* required for the state transition.  *)
    pf_v_output: In (uid voter_output) (get_u_ids_of_snsr_data u_data_lst);
    pf_presrvd_data : presrvd_data.(u_output) = voter_output
                           /\ healthy_data presrvd_data;
    pf_age : output_age = 0 <->
            (voter_validity = valid /\ In presrvd_data u_data_lst);
```

```
pf_validity : pf_validity_prop voter_output voter_validity u_data_lst;
pf_out_not_isolated : voter_validity = valid ->
            In (uid voter_output)
                get_u_ids_of_snsr_data non_isolated_list u_data_lst));
pf_risky_cnt : risky_cnt_age_prop
                  pf_ud_lst pf_v_output output_age voter_validity;
pf_age_bound : voter_validity <> not_valid ->
        forall x,
            In x u_data_lst -> x.(u_status).(iso_status) = not_isolated
              -> output_age - x.(u_status).(risky_count)
                        < persistence_lmt;
pf_age_validity:(output_age < persistence_lmt<->voter_validity = valid)
                 /\ (output_age >= 2*persistence_lmt
                        <-> voter_validity = not_valid)            }.
```

The requirement R6 is maintained as an invariant of `unit_status`. The term `pf_validity` is a direct translation of the requirements R10, R11 and R13. The proof term `pf_risky_cnt` captures the second part of requirement R14. The requirements R8, R15, R16 are also embedded as invariants using other proof terms in the above `voter_state` definition.

Quantum Protocols for Rabin Oblivious Transfer

Erika Andersson
SUPA, Institute of Photonics and Quantum Sciences, School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, UK

Akshay Bansal
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

James T. Peat
SUPA, Institute of Photonics and Quantum Sciences, School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, UK

Jamie Sikora
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA

Jiawei Wu
Centre for Quantum Technologies, National University of Singapore, Singapore

Abstract

Rabin oblivious transfer is the cryptographic task where Alice wishes to receive a bit from Bob but it may get lost with probability 1/2. In this work, we provide protocol designs which yield quantum protocols with improved security. Moreover, we provide a constant lower bound on any quantum protocol for Rabin oblivious transfer. To quantify the security of this task with asymmetric cheating definitions, we introduce the notion of cheating advantage which may be of independent interest in the study of other asymmetric cryptographic primitives.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases quantum cryptography, oblivious transfer, information-theoretic security

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.7

Related Version *Full Version*: https://arxiv.org/abs/2507.04015

Funding *Erika Andersson*: UK Engineering and Physical Sciences Research Council (EPSRC) under Grants No. EP/T001011/1 and EP/Z533208/1.
Jiawei Wu: National Research Foundation, Singapore and A*STAR under its Quantum Engineering Programme (NRF2021-QEP2-01-P06).

Acknowledgements JS thanks Dominique Unruh for pointing out the OT to ROT reduction. AB thanks Or Sattath and Atul Mantri for useful discussions and pointing out the references for protocols under noisy communication.

1 Introduction

Cryptographic tasks where an adversary remains oblivious to the input or the data of other parties are popularly studied under the banner of *oblivious transfer*. Several variants of oblivious transfer tasks exist in the literature today with varying goals and security definitions. A commonly studied variant of oblivious transfer, known as 1-out-of-2 oblivious transfer (OT), is a cryptographic primitive involving two parties, Alice and Bob, where Bob has two bits of information, and Alice wishes to learn one of them based on her choice of an index bit. The security of OT protocols is usually a two-fold evaluation where one needs to protect honest Bob from a cheating Alice who may wish to obtain more information that the protocol allows (perhaps by learning both bits), meanwhile Bob may wish to learn Alice's input, that is, which bit she wishes to learn.


© Erika Andersson, Akshay Bansal, James T. Peat, Jamie Sikora, and Jiawei Wu;
licensed under Creative Commons License CC-BY 4.0

We now define a variant of oblivious transfer, namely 1-out-of-2 OT, with its underlying security notions.

▶ **Definition 1** (1-out-of-2 OT). *1-out-of-2 OT is the cryptographic task between two parties (Alice and Bob) such that*

- *Alice chooses* $a \in \{0,1\}$ *uniformly at random and Bob chooses* $(x_0, x_1) \in \{0,1\}^2$ *uniformly at random.*
- *If both Alice and Bob are honest, then neither Alice nor Bob aborts and Alice outputs* $z = x_a$*.*

We define the following notions of security for protocols that implement 1-out-of-2 OT.

- *Completeness:* If both Alice and Bob are honest, Alice learns the bit x_a unambiguously.
- *Cheating Alice:* A dishonest Alice attempts to learn both x_0 and x_1. The cheating probability of Alice is given by

$$P_A^{OT} = \max \Pr[\text{Alice correctly guesses both } x_0 \text{ and } x_1],$$

  where the maximum is taken over all cheating strategies of Alice. Note that $P_A^{OT} \geq 1/2$ since she can simply follow the protocol to learn x_a perfectly (completeness) and can correctly guess $x_{\overline{a}}$ by making a guess uniformly at random.
- *Cheating Bob:* A dishonest Bob attempts to learn the bit a. The cheating probability of Bob is given by

$$P_B^{OT} = \max \Pr[\text{Bob correctly guesses } a],$$

  where the maximum is taken over all cheating strategies of Bob. Note that $P_B^{OT} \geq 1/2$ since he can correctly guess Alice's bit a by guessing one of the two values uniformly at random.

We note that there are alternative definitions of this task where the inputs are fixed beforehand or possibly generated randomly within the protocol itself. Many of these definitions are equivalent to the one presented above and we refer the reader to [7, 12, 11, 2] for proofs and discussions.

Oblivious transfer, its variants, and other simple cryptographic tasks, are referred to as primitives as they could be used as building blocks for other useful tasks such as coin flipping [22, 25, 3], bit commitment [10], oblivious circuit evaluation [6, 13], and even general multiparty computation [21, 20].

In this work, we also consider the task of coin flipping where Alice and Bob wish to flip a coin but have a different preferred choices for the outcome.

We next discuss coin flipping and provide its security definitions.

▶ **Definition 2** (Coin flipping). *Coin flipping is the cryptographic task between two parties (Alice and Bob) where they communicate to agree on a common binary outcome (*0 *or* 1*). A weak version of this task is a variant where Alice* wins *if the outcome is* 1 *while Bob* wins *if the outcome is* 0*.*

We consider the following notions of security for a given weak coin flipping protocol.

- *Completeness:* If both Alice and Bob are honest, then neither party aborts and the shared outcome is generated uniformly at random.
- *Cheating Alice:* If Bob is honest, then Alice's cheating probability is defined with respect to the outcome $d \in \{0,1\}$ as

$$P_{A,d} = \max \Pr[\text{Alice successfully forces Bob to accept the outcome } d],$$

where the maximum is taken over all possible cheating strategies of Alice. Note that here $P_{A,d} \geq 1/2$ since Alice can simply choose to follow the protocol honestly to observe the outcome d uniformly at random.

- *Cheating Bob:* If Alice is honest, then Bob's cheating probability is defined with respect to the outcome $d \in \{0, 1\}$ as

$$P_{B,d} = \max \Pr[\text{Bob successfully forces Alice to accept the outcome } d],$$

where the maximum is taken over all possible cheating strategies of Bob. As before, here $P_{B,d} \geq 1/2$ since Bob can simply choose to follow the protocol honestly to observe the outcome d uniformly at random.

The variant of the coin flipping task where Alice and Bob do not have any preferred choice of outcome is known as strong coin flipping. We now state Kitaev's lower bound for strong coin flipping which is useful to deduce lower bounds for other two-party primitives.

▶ **Proposition 3** (Kitaev's lower bound [22])**.** For all quantum coin flipping protocols, we have

$$P_{A,d}P_{B,d} \geq 1/2 \tag{1}$$

where $d \in \{0, 1\}$ and $P_{A,d}$ and $P_{B,d}$ are as defined above.

It has been shown that quantum protocols exist for coin flipping up to arbitrary levels of security [25] with explicit constructions given in [3, 4]. This has been generalized to construct secure z-balanced coin flipping in [9] where the two parties output heads with probability z and tails with the remaining probability of $(1 - z)$ securely. Moreover, optimal protocols for the *semi-honest* version of 1-out-of-2 OT were developed in [8], which also provides an the best known lower bound on the security of 1-out-of-2 oblivious transfer as a corollary.

The focus of this work is to examine quantum protocols for a variant of oblivious transfer known as Rabin oblivious transfer. It has been shown that Rabin oblivious transfer is equivalent to 1-out-of-2 oblivious transfer [12], and that 1-out-of-2 oblivious transfer is universal [21]. However, there are only a few papers that studied Rabin oblivious transfer in a quantum setting [5, 33] where the aim is to study its information-theoretic security. The goal of our work is to provide improved bounds on its overall security by finding better protocols as well as lower bounds on how secure they can be.

1.1 Rabin oblivious transfer

In this work, we consider a variant of oblivious transfer, known as *Rabin oblivious transfer*, denoted by ROT. This task was first proposed by Rabin in [29, 30], whence the name. The task was later revisited in [16] to give a protocol that guarantees security under certain computational hardness assumptions. ROT is the cryptographic task which accomplishes the following:

- At the beginning of the protocol, Bob outputs a uniformly random bit $y \in \{0, 1\}$.
- At the end of the protocol, Alice receives y with probability $1/2$, else the bit is lost.

The security of ROT protocols is defined in a two-fold manner. Dishonest Alice wishes to maximize her chances of learning the bit y while dishonest Bob wishes to maximize his chances of successfully guessing whether Alice *asserts* the receipt of the data or the data loss event. Note that if Bob cheats, Alice might think that she learned the bit y, but the learned bit could very well be different from y. Thus, a cheating Bob only cares if Alice *thinks* she

correctly learned y. Note that in contrast to the analysis described in [16], we work under the regime of information-theoretic security where the adversaries are not bounded under any computational assumptions.

We now provide a more formal definition.

▶ **Definition 4** (Rabin OT). *Rabin OT (ROT) is the cryptographic task between two parties (Alice and Bob) such that:*

- *Bob chooses $y \in \{0,1\}$ uniformly at random.*
- *At the end of the protocol, Alice outputs a bit $g \in \{0,1\}$ with probability $1/2$ (which corresponds to her asserting that she learned the bit y) or NULL with probability $1/2$ (which corresponds to her losing the bit).*

The completeness and security notions for an ROT protocol are below.

- *Completeness:* If both Alice and Bob are honest, neither party aborts. Alice outputs $g = y$ with probability $1/2$ or NULL with probability $1/2$.
- *Cheating Alice:* If Bob is honest, then dishonest Alice's cheating probability is defined as

$$P_A^* = \max\{\Pr[\text{Alice successfully learns } y]\},$$

where the maximum is taken over all cheating strategies of Alice. Note that $P_A^* \geq 3/4$ as Alice can simply choose to follow the protocol to learn y with probability $1/2$ and randomly guess y in the event of NULL.

- *Cheating Bob*: If Alice is honest, then dishonest Bob's cheating probability is defined as

$$P_B^* = \max\{\Pr[\text{Bob correctly guesses whether Alice } \textit{asserts} \text{ the receipt of } y \text{ or NULL}]\},$$

where the maximum is taken over all cheating strategies of Bob. More formally, if we denote Alice's assertion by bit a, then $a = 0$ denotes that Alice asserted the receipt of y, while $a = 1$ denote that she asserted the receipt of NULL. Here, P_B^* could be alternatively expressed as the maximum probability with which Bob correctly learns a.
We emphasize that Alice asserting (possibly privately) the receipt of y (by outputting g) does not necessarily imply the receipt of the actual y, i.e., it could be the case that $g \neq y$, or even that y does not even exist. Note that $P_B^* \geq 1/2$ as Bob can simply choose to follow the protocol honestly and make a uniformly random guess to whether Alice asserts y or NULL.

As the cheating notions of Alice and Bob in the ROT are asymmetric, especially with respect to the lower bounds of $3/4$ and $1/2$ on their respective cheating probabilities, we define a single security measure of how good an ROT protocol is with respect to its pair of cheating probabilities (P_A^*, P_B^*). To achieve this, we introduce the concept of the *cheating advantage* of a protocol, denoted as γ, defined as:

$$\begin{aligned} \gamma &:= \max\{\gamma_A, \gamma_B\}, \\ \gamma_A &:= P_A^*/P_A^{ideal} \quad \text{where} \quad P_A^{ideal} = 3/4, \quad \text{and} \\ \gamma_B &:= P_B^*/P_B^{ideal} \quad \text{where} \quad P_B^{ideal} = 1/2. \end{aligned} \tag{2}$$

The ultimate objective of designing ROT protocols is thus to find one which *minimizes the cheating advantage*, with the hopeful goal of making γ as close to 1 as possible. (We note that $\gamma \leq 2$ since both P_A^* and P_B^* are no greater than 1.)

We remark that this notion of *cheating advantage* is similar to what is done in [11] for the case of k-out-of-n (forcing) oblivious transfer which also has vastly different ideal cheating probabilities for Alice and Bob. We believe that this is a natural security measure that can

be adopted by other primitives, even symmetric ones. However, since we are dealing with two cheating parameters, it could be the case that some applications would desire to minimize a different function of these two probabilities, different than the cheating advantage. While we do look at cheating advantage in this work, we note that our focus is on the design of protocols that minimize both the cheating probabilities, and the cheating advantage is just a way to assign a quality measure to a given protocol.

1.2 Prior work on quantum protocols for Rabin oblivious transfer and related cryptographic tasks

A first quantum ROT protocol with non-trivial security guarantees was given in [5] where a couple different variants in the security definition were considered including the one described in this work. Specifically, the authors first developed a framework of stochastic switching and then later used it to combine a couple of different insecure Rabin OT protocols to yield one with better security. This protocol yields $\gamma_A = 1.29$ and $\gamma_B = 1.87$, implying $\gamma = 1.87 < 2$ for the security definition considered in Section 1.1.

In [33], a quantum ROT protocol using two pure states is investigated, and shown to outperform classical protocols in some parameter regimes. Also, it was previously claimed that there exists a perfectly secure quantum protocol for Rabin OT [18]. However, this protocol turned out to be vulnerable to cheating [28]. In fact, a constant lower bound on the overall security of a variant of ROT in the setting of secure function evaluation (with additional inputs) can be shown from the general lower bound in [27] and is given explicitly in the thesis [26, Section 4.7], hinting at the existence of general lower bounds for ROT.

If one makes no assumptions on the cheating powers of Alice and Bob, for example, that they are computationally bounded, the limits on the unconditional security for 1-out-of-2 OT was studied in [11] which provided a constant lower bound on the worst-case security for all complete[1] protocols. This result extends the no-go result in [23] that shows that general quantum computations cannot be done securely, which includes the special case of OT. The lower bound was later improved in [8] which showed that a variant of OT, known as *semi-honest* oblivious transfer, where dishonest Alice is restricted to the set of strategies that allows her to learn the chosen bit perfectly.

More recently, a lower bound on the security of 1-out-of-2 OT [2] has been used to deduce fundamental trade-offs between circuit privacy, data privacy and correctness for a broad family of quantum homomorphic encryption protocols [19].

In terms of finding protocols, it was shown in [11] that partial security for the 1-out-of-2 OT task can be obtained, that is, the maximum probability with which Alice or Bob could cheat successfully is strictly below 1, a strict (quantum) improvement over classical protocols where it is known that either Alice or Bob can always cheat perfectly. Finding optimal protocols for 1-out-of-2 OT is still an interesting, and important, open problem.

1.3 A few remarks about the security setting considered in this work

Here we discuss how our study of Rabin oblivious transfer fits into the bigger picture of two-party quantum cryptography.

First and foremost, we are considering *information-theoretic* security, where we do not place any restriction on a cheating party. For example, we do not make any assumptions on computational hardness [37, 17], space limitations [31, 15], noisy hardware [35, 32], etc.

[1] Complete protocols are defined such that honest participants achieve the desired objective.

While these are interesting in their own right, and can be well-motivated, we are studying this task in the most demanding setting possible, that is, when Alice and Bob are only bounded by the laws of quantum mechanics.

Secondly, we are assuming that all communication channels are noiseless as the problem could become easier in some noisy settings. Consider a binary erasure channel which transmits a bit x and, with probability p, the bit is replaced with a flag $\perp$. Then for $p = 1/2$, this can accomplish exactly Rabin oblivious transfer. Moreover, Alice and Bob cannot cheat in this setting. *In this sense, Rabin oblivious transfer seeks to emulate a binary erasure channel where the noise is controlled by Alice or Bob, not the environment.* Indeed, it is possible to efficiently implement other tasks such as bit commitment and oblivious transfer in the noisy setting [14].

Lastly, we would like to emphasize that unlike some of the previous works that studied the composability of various two-party tasks such as coin flipping [34, 36], here we only consider the standalone security of the primitives relevant to this work, including Rabin oblivious transfer, 1-out-of-2 oblivious transfer and coin flipping. Therefore, the protocols we propose in the later sections of this work may not be necessarily secure under arbitrary compositions, and we leave this research direction for future work.

1.4 New protocols for Rabin oblivious transfer

In this section, we present novel constructions of Rabin oblivious transfer protocols that offer a strict improvement in overall security compared to those described in previous works [5, 33]. Here, we adopt the security notion discussed in Section 1.1.

1.4.1 A quantum protocol via weak coin flipping

One way to reduce the cheating advantages of many two-party cryptographic protocols is to balance them using *weak coin flipping*. Suppose that you are given two protocols *for the same task* $\mathcal{P}_1$ with $\mathcal{P}_2$ and suppose that Alice can cheat more in $\mathcal{P}_1$ than in $\mathcal{P}_2$ and Bob can cheat more in $\mathcal{P}_2$ than in $\mathcal{P}_1$. In other words, if Alice and Bob were faced with deciding on whether to use $\mathcal{P}_1$ or $\mathcal{P}_2$, then they would want opposing decisions. One way to deal with this is to have them *flip a coin*, and if they output HEADS then use $\mathcal{P}_1$, else use $\mathcal{P}_2$. This also has the potential benefit of reducing the overall cheating, as described in the following lemma.

▶ **Lemma 5** (Balancing Lemma)**.** *Suppose we are given two quantum ROT protocols with $\mathcal{P}_1$ with cheating advantages $(\gamma_A^{(1)}, \gamma_B^{(1)})$ and $\mathcal{P}_2$ with cheating advantages $(\gamma_A^{(2)}, \gamma_B^{(2)})$. If Alice prefers $\mathcal{P}_1$ (i.e., $\gamma_A^{(1)} \geq \gamma_A^{(2)}$), Bob prefers $\mathcal{P}_2$ (i.e., $\gamma_B^{(1)} \leq \gamma_B^{(2)}$), and $0 < \frac{\gamma_B^{(2)} - \gamma_A^{(2)}}{\gamma_B^{(2)} - \gamma_A^{(2)} + \gamma_A^{(1)} - \gamma_B^{(1)}} < 1$ then there exists a family of quantum ROT protocols approaching*

$$\gamma = \gamma_A = \gamma_B = \frac{\gamma_A^{(1)}\gamma_B^{(2)} - \gamma_B^{(1)}\gamma_A^{(2)}}{\gamma_B^{(2)} - \gamma_A^{(2)} + \gamma_A^{(1)} - \gamma_B^{(1)}} \leq \min\{\gamma_1, \gamma_2\}, \tag{3}$$

where $\gamma_1 = \max\{\gamma_A^{(1)}, \gamma_B^{(1)}\}$ and $\gamma_2 = \max\{\gamma_A^{(2)}, \gamma_B^{(2)}\}$. The improvement is strict, i.e., we have $\gamma < \min\{\gamma_1, \gamma_2\}$ under the previous conditions if and only if Alice has a strict preference for $\mathcal{P}_1$ ($\gamma_A^{(1)} > \gamma_A^{(2)}$) and Bob strictly prefers $\mathcal{P}_2$ ($\gamma_B^{(1)} < \gamma_B^{(2)}$).

▶ Remark 6. This balancing lemma above can in general be used for any two protocols for a given two-party task (coin flipping, bit commitment, etc.) to potentially yield another protocol with improved security. We note that this trick uses a quantum protocol for coin flipping for the information-theoretic security setting. In other settings, one must be mindful of the coin flipping protocol security when balancing in this way.

We note that if both Alice and Bob prefer the same protocol over the other, e.g., $\gamma_A^{(1)} \leq \gamma_A^{(2)}$ and $\gamma_B^{(1)} \leq \gamma_B^{(2)}$, then this balancing lemma does not offer any advantages. However, its power comes from the ability to combine protocols for which Alice and Bob have different preferences.

Proof Sketch. The proof follows immediately using the task of *unbalanced weak coin flipping* for which optimal quantum protocols are known [9], based on optimal weak (balanced) coin flipping protocols [25, 3]. Specifically, for the z-unbalanced weak coin flipping task between Alice and Bob, for all choices of $\epsilon > 0$, there exist a sequence of protocols which in the limit have the following features:

- The probability that the outcome is HEADS is z when Alice and Bob are honest, and they both output the same coin flip;
- Alice cannot force the outcome HEADS with probability greater than z;
- Bob cannot force the outcome TAILS with probability greater than $1 - z$.

Suppose Alice and Bob use a z-unbalanced weak coin flipping protocol with $z = \frac{\gamma_B^{(2)} - \gamma_A^{(2)}}{\gamma_B^{(2)} - \gamma_A^{(2)} + \gamma_A^{(1)} - \gamma_B^{(1)}}$, then on observing the outcome HEADS, they perform $\mathcal{P}_1$ and on outcome TAILS then perform $\mathcal{P}_2$. The calculation of γ_A or γ_B follows directly by calculating $z\gamma_A^{(1)} + (1-z)\gamma_A^{(2)}$ or $z\gamma_B^{(1)} + (1-z)\gamma_B^{(2)}$ for $z = \frac{\gamma_B^{(2)} - \gamma_A^{(2)}}{\gamma_B^{(2)} - \gamma_A^{(2)} + \gamma_A^{(1)} - \gamma_B^{(1)}}$. ◀

This lemma turns out to be valuable for minimizing the cheating advantage in the protocols in this paper, mostly due to the fact that Alice and Bob's cheating probabilities are rather imbalanced. But before that, we demonstrate that this simple act of balancing can turn bad protocols into decent protocols.

Consider the following two *bad* protocols for ROT.

▶ **Protocol 1.** A bad Rabin oblivious transfer protocol $\mathcal{P}_1$.

- Bob chooses $y \in \{0, 1\}$ uniformly at random.
- Bob flips a fair coin.
  - If HEADS, Bob announces y to Alice.
  - If TAILS, Bob announces "NULL" to Alice.

Clearly this protocol satisfies $P_A^* = 3/4$ since Alice either learns y or not, and if not she has to guess. Moreover, $P_B^* = 1$ as Bob knows exactly what Alice would assert. Thus, for this protocol we have

$$\gamma_A = \frac{3/4}{3/4} = 1, \quad \gamma_B = \frac{1}{1/2} = 2, \quad \text{and} \quad \gamma = \max\{\gamma_A, \gamma_B\} = 2, \tag{4}$$

which is the worst value of γ possible.

▶ **Protocol 2.** Another bad Rabin oblivious transfer protocol $\mathcal{P}_2$.

- Bob chooses $y \in \{0, 1\}$ uniformly at random.
- Bob prepares $|y\rangle$ and sends it to Alice.
- Alice flips a fair coin.
  - If HEADS, Alice measures $|y\rangle$ to learn y.
  - If TAILS, Alice throws the qubit away and accepts NULL as her outcome.

Clearly this protocol satisfies $P_A^* = 1$ since she can simply measure the qubit to learn it perfectly. Moreover, $P_B^* = 1/2$, since Alice has full control over what she asserts, and no information is sent to Bob after Alice decides to measure or not. Thus, for this protocol we have

$$\gamma_A = \frac{1}{3/4} = \frac{4}{3}, \quad \gamma_B = \frac{1/2}{1/2} = 1, \quad \text{and} \quad \gamma = \max\{\gamma_A, \gamma_B\} = \frac{4}{3}, \tag{5}$$

which is not the worst value of γ possible, but still not that good.

Even though protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ have poor security, we can apply the Balancing Lemma to improve their overall security resulting in the following theorem.

▶ **Theorem 7.** *There exists a quantum protocol for ROT with cheating advantage $\gamma = 5/4$.*

A few remarks are in order. Note that even though this protocol is easy to describe, just use optimal unbalanced weak coin flipping with the two bad ROT protocols, it is important to note that optimal unbalanced weak coin flipping protocols are very complicated. The only way we know how to construct such protocols is via optimal weak coin flipping [25]. Despite much work done to simplify such protocols [1, 3], they are still very technical. Moreover, it has been shown that such protocols cannot be efficient [24]. Thus, this begs the question:

Does there exist ROT protocols that are simple and offer decent security?

In the following two sections, we describe how to achieve this.

1.4.2 A quantum protocol via a reduction from oblivious transfer

Typically, reductions between tasks are useful to establish relationships between them, which may include non-trivial security bounds [11]. In some cases, reductions can also be used to yield protocols with desirable security. However, some of the fundamental security notions, such as *completeness* might be compromised while reducing one task to another. Here, it is crucial to devise appropriate reductions for tasks where no such violations are permitted.

We next depict a folklore (classical) reduction from OT to ROT. Alice and Bob communicate only classically outside the OT subroutine to perform the ROT task. We characterize this as a classical reduction given that the communication outside of the OT subroutine is classical.

▶ **Reduction 1.** Rabin oblivious transfer from 1-out-of-2-oblivious transfer.

- Bob chooses data $y \in \{0, 1\}$ and a dummy bit $z \in \{0, 1\}$ uniformly at random.
- Bob chooses a permutation value $p \in \{0, 1\}$ uniformly at random and assigns the bits $(x_0, x_1) = (y, z)$ if $p = 0$, or $(x_0, x_1) = (z, y)$, otherwise.
- **Alice and Bob perform 1-out-of-2 oblivious transfer** so that Alice learns either x_0 or x_1 (see details on 1-out-of-2 OT in Definition 1).
- Bob sends p to Alice. Alice now knows whether she learned y or z.
- If she learned z, then she outputs NULL. Otherwise, she outputs y.

It is straightforward to observe that an ROT implementation using Reduction 1 is *complete* whenever the underlying OT protocol is *complete*. Additionally, the reduction is tight in the sense that given a perfectly secure OT protocol the resultant ROT protocol also enjoys perfect security. Of course, perfectly secure OT is impossible as discussed earlier, thus this reduction does not give us perfectly secure ROT protocols. However, it does give partial security, as discussed below.

Using this reduction, our idea to create ROT protocols is now clear, we can simply choose an existing (or develop a new) quantum OT protocol and slot it into Reduction 1. However, since this is a quantum protocol, it is *not clear* how Alice and/or Bob may cheat with regards to composing various cheating strategies. That is, an optimal cheating strategy in the OT protocol may not be the optimal cheating strategy in the overall protocol. This optimal cheating behavior (and thus the worst case cheating probabilities) needs to be proven for this reduction, since we do not know much about the security aspects of 1-out-of 2 oblivious transfer protocols when used as a subroutine in a larger composition. Thus, we provide a rigorous self-contained security analysis of the Rabin OT protocols resulting from the previous reduction.

We now exhibit a complete quantum protocol for ROT, below.

▶ **Protocol 3.** Using Reduction 1 with the 1-out-of-2 OT protocol from [11].

- Alice chooses a bit $a \in \{0,1\}$ uniformly at random and creates the two-qutrit state

$$|\phi_a\rangle = \frac{1}{\sqrt{2}}|aa\rangle + \frac{1}{\sqrt{2}}|22\rangle \in \mathcal{A}\otimes\mathcal{B}.$$

- Alice sends the qutrit $\mathcal{B}$ to Bob.
- Bob chooses data bit $y \in \{0,1\}$ and $z \in \{0,1\}$ uniformly at random.
- Bob chooses a permutation bit $p \in \{0,1\}$ uniformly at random and assigns $(x_0, x_1) = (y, z)$ if $p = 0$, or $(x_0, x_1) = (z, y)$, otherwise.
- Bob applies the unitary

$$U_{x_0x_1} = \begin{bmatrix} (-1)^{x_0} & 0 & 0 \\ 0 & (-1)^{x_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  to $\mathcal{B}$ and sends it back to Alice.
- Alice determines the value of x_a using the two-outcome measurement:

$$\{\Pi_0 \coloneqq |\phi_a\rangle\langle\phi_a|_{\mathcal{A}\otimes\mathcal{B}}, \Pi_1 \coloneqq \mathbb{1}_{\mathcal{A}\otimes\mathcal{B}} - |\phi_a\rangle\langle\phi_a|\}.$$

- Bob sends p to Alice.
- If $a = p$, Alice asserts the receipt of $y = x_a$, else she asserts NULL.

The completeness of the 1-out-of-2 protocol in [11] ensures that this ROT protocol is also *complete*. However, it is not clear whether the optimal cheating probability for dishonest Alice where she tries to learn Bob's data bit y is equivalent to Alice learning both x_0 and x_1 in the reduced 1-out-of-2 OT protocol. Although learning just these two bits (x_0 and x_1) qualifies as a valid cheating strategy, she might deploy a more clever strategy to improve her chances of successfully learning y, i.e., we have $P_A^* \geq P_A^{OT}$. Note that quantum communication in the reduced 1-out-of-2 OT protocol attributes to a non-trivial overall analysis as dishonest Alice could now correlate her messages with Bob's input data. We give an argument of Alice's possible strategies below, and provide the missing details in the appendix.

A general strategy for a cheating Alice has her creating a state of the form below

$$|\psi'\rangle = \alpha|\psi_\alpha\rangle_{\mathcal{A}'}|0\rangle_{\mathcal{B}} + \beta|\psi_\beta\rangle_{\mathcal{A}'}|1\rangle_{\mathcal{B}} + \gamma|\psi_\gamma\rangle_{\mathcal{A}'}|2\rangle_{\mathcal{B}},$$

where $\alpha, \beta, \gamma \geq 0$ satisfy $\alpha^2 + \beta^2 + \gamma^2 = 1$, and sends the subsystem $\mathcal{B}$ to Bob. Once Bob applies the unitary $U_{x_0x_1}$ on the subsystem $\mathcal{B}$ and sends it back, Alice holds the state

$$|\psi'_{x_0x_1}\rangle = \alpha(-1)^{x_0}|\psi_\alpha\rangle_{\mathcal{A}'}|0\rangle_{\mathcal{B}} + \beta(-1)^{x_1}|\psi_\beta\rangle_{\mathcal{A}'}|1\rangle_{\mathcal{B}} + \gamma|\psi_\gamma\rangle_{\mathcal{A}'}|2\rangle_{\mathcal{B}}.$$

It is important to observe that Alice's choice of the states $\{|\psi_\alpha\rangle, |\psi_\beta\rangle, |\psi_\gamma\rangle\}$ is not relevant in maximizing her chances of successfully cheating in Protocol 3 as for any choice of $|\psi_\alpha\rangle$, $|\psi_\beta\rangle$ and $|\psi_\gamma\rangle$, there exists a unitary U such that

$$U|\psi'\rangle = |\psi\rangle,$$

where $|\psi\rangle = \alpha|0\rangle_{\mathcal{A}'}|0\rangle_{\mathcal{B}} + \beta|0\rangle_{\mathcal{A}'}|1\rangle_{\mathcal{B}} + \gamma|0\rangle_{\mathcal{A}'}|2\rangle_{\mathcal{B}}$.
Thus, Alice could restrict to creating states of the form $|\psi\rangle$, or equivalently just the states of the form

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle \in \mathcal{B}.$$

Subsequently, her task reduces to extracting x_0 or x_1 (depending on p) from the reduced state

$$|\psi_{x_0x_1}\rangle = \alpha(-1)^{x_0}|0\rangle + \beta(-1)^{x_1}|1\rangle + \gamma|2\rangle \in \mathcal{B}. \tag{6}$$

If $p = 0$, the probability of Alice successfully guessing y (or x_0 in this case) is given by the Holevo-Helstrom theorem as

$$\Pr[\text{Alice successfully learns } y|p=0] = \frac{1}{2} + \frac{1}{4}\left\|\rho_0(0) - \rho_1(0)\right\|_1,$$

where $\rho_0(0) = \frac{1}{2}|\psi_{00}\rangle\langle\psi_{00}| + \frac{1}{2}|\psi_{01}\rangle\langle\psi_{01}|$ and $\rho_1(0) = \frac{1}{2}|\psi_{10}\rangle\langle\psi_{10}| + \frac{1}{2}|\psi_{11}\rangle\langle\psi_{11}|$.
Similarly, if $p = 1$, the probability of Alice successfully guessing y (or x_1 in this case) is given by

$$\Pr[\text{Alice successfully learns } y|p=1] = \frac{1}{2} + \frac{1}{4}\left\|\rho_0(1) - \rho_1(1)\right\|_1,$$

where $\rho_0(1) = \frac{1}{2}|\psi_{00}\rangle\langle\psi_{00}| + \frac{1}{2}|\psi_{10}\rangle\langle\psi_{10}|$ and $\rho_1(1) = \frac{1}{2}|\psi_{01}\rangle\langle\psi_{01}| + \frac{1}{2}|\psi_{11}\rangle\langle\psi_{11}|$.
Optimizing the overall success probability over all choices of α, β and γ, we get $P_A^* = \cos^2(\pi/8)$ when $\alpha = \beta = 1/2$ and $\gamma = 1/\sqrt{2}$.

We now state the overall security of this protocol and defer the rest of the proof to Appendix A.

▶ **Lemma 8.** *The ROT Protocol 3 has cheating probabilities $P_A^* = \cos^2(\pi/8)$ and $P_B^* = 3/4$ implying $\gamma_A = 1.138$ and $\gamma_B = 1.5$, resulting in a cheating advantage of $\gamma = 1.5$.*

By combining Protocol 3 with Protocol 2 using the Balancing Lemma 5, we get an ROT protocol, albeit inefficient due to [24], with security strictly better than the protocol discussed in Section 1.4.1 and is stated in the following theorem.

▶ **Theorem 9.** *There exists a quantum protocol for ROT with cheating advantage $\gamma = 1.239$.*

We now describe a way to execute many ROT protocols in succession using a simpler protocol which exhibits the same cheating probabilities as derived previously for Protocol 3.

1.5 A simple sequence of Rabin oblivious transfer protocols

We now propose another way of performing Rabin oblivious transfer based on the OT protocols described in [2]. We describe the protocol below.

▶ **Protocol 4.** A sequence of Rabin oblivious transfers.

- For each $i \in \{1, \ldots, N\}$, Bob picks a pair of bits (x_0^i, x_1^i), all independently and uniformly at random.

- For each $i \in \{1, \ldots, N\}$, Bob prepares the two-qubit states $|\Psi_{x_0^i x_1^i}\rangle$, where we define:

$$|\Psi_{00}\rangle = |00\rangle\,, \quad |\Psi_{01}\rangle = |{+}{+}\rangle\,, \quad |\Psi_{10}\rangle = |{-}{-}\rangle\,, \quad |\Psi_{11}\rangle = |11\rangle\,, \tag{7}$$

  and $|\pm\rangle := (|0\rangle \pm |1\rangle)/\sqrt{2}$. Bob sends the states to Alice.
- Alice selects a subset S of the set $\{1, \ldots, N\}$ of size $\left\lfloor \sqrt{N} \right\rfloor$ uniformly at random. For each $j \in S$, Alice asks Bob to announce the state $|\Psi_{x_0^j, x_1^j}\rangle$.
- If Bob announces $|{+}{+}\rangle$ or $|{-}{-}\rangle$, then Alice measures both qubits in the $\{|+\rangle, |-\rangle\}$ basis, otherwise Alice measures in the $\{|0\rangle, |1\rangle\}$ basis. Alice aborts if any of her outcomes are not consistent with Bob's purported states. If Alice does not abort, then the $\left\lfloor \sqrt{N} \right\rfloor$ states used for these tests are discarded.
- For each $j \notin S$, Alice measures the first qubit of $|\Psi_{x_0^j x_1^j}\rangle$ in the $\{|0\rangle, |1\rangle\}$ basis and the second qubit of $|\Psi_{x_0^j x_1^j}\rangle$ in the $\{|+\rangle, |-\rangle\}$ basis. This way Alice learns one of Bob's bits with certainty while gaining no knowledge of the other (each of her measurement outcomes allows her to rule out one of Bob's possible states). For example, if her measurement outcome is
  - 00: then we have $(x_0^j, x_1^j) = (0,0)$ or $(0,1)$, i.e., the first bit is 0;
  - 01: then we have $(x_0^j, x_1^j) = (0,0)$ or $(1,0)$, i.e., the second bit is 0;
  - 10: then we have $(x_0^j, x_1^j) = (0,1)$ or $(1,1)$, i.e., the second bit is 1;
  - 11: then we have $(x_0^j, x_1^j) = (1,0)$ or $(1,1)$, i.e., the first bit is 1.
- For each $j \notin S$, Bob declares which of the two bits will be used for Rabin oblivious transfer. This is chosen uniformly at random for each j.

As before, the completeness follows from the completeness of the underlying OT protocol. Again we have to carefully consider what a cheating Alice can do to further her cheating probability over what she can achieve if she uses the cheating strategy which is optimal in the OT protocol.

The optimal cheating probabilities for Alice and Bob are given in the following lemma.

▶ **Lemma 10.** *The ROT Protocol 4 has cheating probabilities $P_A^* = \cos^2(\pi/8)$ for all N and $P_B^* = 3/4$ in the limit $N \to \infty$ implying $\gamma_A = 1.138$ and $\gamma_B = 1.5$, resulting in a cheating advantage of $\gamma = 1.5$ in the limit.*

Proof. In order to prove Lemma 10, we separately analyze the security of the protocol against cheating Alice and Bob as discussed next.

Cheating Alice. In this protocol Alice only receives states, and sends nothing to Bob. This means that the only cheating method available to Alice is to use a measurement which is different from what she would measure if honest. As Bob decides which of the two bits that counts for each of the remaining $N - \left\lfloor \sqrt{N} \right\rfloor \approx N$ states, Alice's best strategy is to store these states until Bob declares which bit value counts. Alice then makes a minimum-error measurement individually on each state, aiming to learn the bit that Bob chose. Due to symmetry, Alice's probability of correctly obtaining the first bit is the same as for the second. When Bob declares that the first bit should be used, the two states Alice has to distinguish between are given by

$$\rho_0 = \frac{1}{2}(|00\rangle\langle 00| + |{+}{+}\rangle\langle {+}{+}|), \quad \rho_1 = \frac{1}{2}(|11\rangle\langle 11| + |{-}{-}\rangle\langle {-}{-}|). \tag{8}$$

Here Alice's best strategy is to make a minimum-error measurement on each instance individually. Since there are only two states to distinguish between, we use the standard result for the probability of success, the so-called Holevo-Helstrom measurement, yielding

$$p_s = \frac{1}{2} + \frac{1}{4}\|\rho_0 - \rho_1\|_1, \tag{9}$$

where $|A| = \sqrt{A^\dagger A}$. This gives Alice's cheating probability as

$$P_A^* = \frac{1}{4}(2 + \sqrt{2}) \approx 0.853 \tag{10}$$

for all choices of N.

Cheating Bob. A cheating Bob in the 1-out-of-2 protocol wants to know which bit Alice received. This is exactly what he needs to know whether or not Alice has received the bit value in the Rabin protocol we have constructed as well. Therefore, Bob's cheating probability is

$$P_B^* = \frac{3}{4}, \tag{11}$$

which follows directly from [2], and is only valid in the limit as $N \to \infty$. This limit assures the fraction tested becomes negligible disallowing further cheating possibilities for Bob. ◀

1.6 A constant lower bound on γ for any quantum protocol for Rabin oblivious transfer

The two main objectives for understanding the limitations of the security for any cryptographic task is to (a) find good protocols and the corresponding security measures, and (b) find lower bounds on such security measures. By improving either of these objectives brings us that much closer to finding the *best* protocol.

So far, we have been concentrating on objective (a), finding good quantum protocols. Now, we consider objective (b), finding lower bounds.

Why care about lower bounds?

Suppose one were to show a constant lower bound on γ, that is, suppose $\gamma \geq c$ for some constant $c > 1$. This implies two things. First, this means that obviously one cannot find protocols with $\gamma < c$, which could suggest that any existing protocols are better than expected (by being now closer to the optimal γ). Secondly, and importantly, this *rules out security amplification attempts*. One may wonder if ROT can be repeated many times and then somehow by putting all the pieces together reduce the cheating probabilities to their ideal goals. Having a constant lower bound rules out such attempts, since near-perfect protocols cannot exist, even in the limit of a family of protocols. Thus, just by having a constant lower bound, regardless of the constant, provides insight towards how one may design optimal protocols. We therefore ask the question:

Does there exist a constant lower bound for Rabin oblivious transfer?

At first glance, it is not always obvious how to guess this. For example, quantum protocols for strong coin flipping (defined shortly) have a constant lower bound [22] while quantum protocols for weak coin flipping do not [25]. What about Rabin oblivious transfer? In this section, we answer this question in the affirmative.

1.6.1 Strong coin flipping

We have discussed (unbalanced) weak coin flipping between Alice and Bob, the task of strong coin flipping is similar. Roughly, a strong coin flipping protocol is defined such that:

- The probability that the outcome is HEADS is 1/2 when Alice and Bob are honest, and they both output the same coin flip;
- Alice can force the outcome HEADS with probability $P_{A,0}$;
- Alice can force the outcome TAILS with probability $P_{A,1}$;
- Bob can force the outcome HEADS with probability $P_{B,0}$;
- Bob can force the outcome TAILS with probability $P_{B,1}$.

A formal definition is given in Definition 2. While much is now known about the limits of the achievable values of $P_{A,0}$, $P_{A,1}$, $P_{B,0}$, $P_{B,1}$, in this discussion we only require one bound, proven by Kitaev [22] (see Proposition 3):

$$P_{A,0} \cdot P_{B,0} \geq 1/2. \tag{12}$$

In other words, in *any* coin flipping protocol, either Alice or Bob can force the outcome 0 with probability at least $1/\sqrt{2} > 1/2$.

1.6.2 A reduction from Rabin oblivious transfer to strong coin flipping

We present our reduction below.

▶ **Protocol 5.** Coin flipping from Rabin oblivious transfer

- Alice and Bob perform an ROT protocol with Bob's data bit denoted by y. We denote Alice's ROT output by the trit $\hat{g} \in \{0, 1, NULL\}$. If Alice asserts the receipt of data, then $\hat{g} = y$, else $\hat{g} = NULL$.
- Alice defines the assert bit $a \in \{0, 1\}$ to be 0 if $\hat{g} \in \{0, 1\}$, and to be 1 if $\hat{g} = NULL$.
- Bob chooses b uniformly at random and sends b to Alice.
- Alice sends a and $\hat{g}$ to Bob. Bob checks if $(a, y, \hat{g})$ are consistent, i.e., he checks if $\hat{g} = y$ when $a = 0$. If $\hat{g} \neq y$, Bob aborts.
- If Alice and Bob do not abort, then they both output HEADS if $a = b$, and TAILS otherwise.

We first remark that this protocol is complete, i.e., both Alice and Bob output the same coin flip whose value is generated uniformly at random. Below, we investigate the security of this coin flipping protocol with respect to the security of the ROT protocol.

▶ **Lemma 11.** *In Protocol 5, we have*

$$P_{A,0} = \frac{1}{2} + \frac{1}{2}P_A^* \quad \text{and} \quad P_{B,0} = P_B^*, \tag{13}$$

where P_A^* and P_B^* are the cheating probabilities in the ROT subroutine.

Proof. Since dishonest Alice wants the outcome HEADS, she must send back $a = b$ in the last message. However, this is only accepted if $(a, y, \hat{g})$ are consistent. If $b = 0$, then Alice must return $a = 0$, in which case Bob accepts the value of a if $\hat{g} = y$. The maximum probability that Alice can pass the test in this case is the maximum probability that she can learn y in the ROT protocol, which is precisely P_A^*. If $b = 1$, then Alice must return $a = 1$, but in this case, Bob is expecting $\hat{g} = NULL$, and thus if she sends this, Bob has nothing to check, and will simply accept the value of a. Since b is chosen independently from the rest of the protocol, we have that $P_{A,0} = \frac{1}{2} + \frac{1}{2}P_A^*$, as desired.

On the other hand, since dishonest Bob want HEADS, he must infer the value of a ahead of time when he sends b. In other words, he must send $b = a$. Therefore, he is successful in forcing Alice to output HEADS with the same probability with which he can learn a in the protocol. Since the value of a corresponds to exactly whether or not Alice asserts the value of Bob's bit y, we have that the maximum probability with which Bob learns a in the coin flipping protocol is simply P_B^*.

By combining Equations (2) and (12), i.e., that

$$\gamma = \max\{\gamma_A, \gamma_B\}, \quad \gamma_A = \frac{P_A^*}{3/4}, \quad \gamma_B = \frac{P_B^*}{1/2}, \quad \text{and} \quad P_{A,0} \cdot P_{B,0} \geq 1/2 \tag{14}$$

together with Lemma 11, one gets the quadratic inequality

$$\frac{1}{2}\gamma\left(\frac{1}{2} + \frac{1}{2}\cdot\frac{3}{4}\gamma\right) \geq \frac{1}{2}, \tag{15}$$

implying that $\gamma \geq \frac{2}{3}\left(\sqrt{7} - 1\right) > 1$. ◀

Therefore, we have the following theorem.

▶ **Theorem 12.** *Any quantum protocol for Rabin oblivious transfer has a cheating advantage satisfying* $\gamma \geq \frac{2}{3}\left(\sqrt{7} - 1\right) \approx 1.097 > 1$.

By combining Theorems 9 and 12, we get the following corollary.

▶ **Corollary 13.** *The optimal quantum protocol for Rabin oblivious transfer has cheating advantage* $\gamma \in [1.097, 1.239]$.

References

1. Dorit Aharonov, André Chailloux, Maor Ganz, Iordanis Kerenidis, and Loïck Magnin. A simpler proof of the existence of quantum weak coin flipping with arbitrarily small bias. *SIAM Journal on Computing*, 45(3):633–679, 2016. doi:10.1137/14096387X.
2. Ryan Amiri, Robert Stárek, David Reichmuth, Ittoop V. Puthoor, Michal Mičuda, Ladislav Mišta, Jr., Miloslav Dušek, Petros Wallden, and Erika Andersson. Imperfect 1-Out-of-2 Quantum Oblivious Transfer: Bounds, a Protocol, and its Experimental Implementation. *PRX Quantum*, 2:010335, 2021.
3. Atul Singh Arora, Jérémie Roland, and Chrysoula Vlachou. Analytic quantum weak coin flipping protocols with arbitrarily small bias. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 919–938, 2021. doi:10.1137/1.9781611976465.58.
4. Atul Singh Arora, Jérémie Roland, Chrysoula Vlachou, and Stephan Weis. Protocols for quantum weak coin flipping. arXiv preprint arXiv:2402.15855, 2024. doi:10.48550/arXiv.2402.15855.
5. Akshay Bansal and Jamie Sikora. Breaking barriers in two-party quantum cryptography via stochastic semidefinite programming. *Quantum*, 9:1602, 2025. doi:10.22331/Q-2025-01-20-1602.
6. Charles H. Bennett, Gilles Brassard, Claude Crépeau, and Marie-Hélène Skubiszewska. Practical quantum oblivious transfer. In *Advances in Cryptology — CRYPTO '91*, pages 351–366, 1992.
7. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Proceedings on Advances in Cryptology—CRYPTO '86*, pages 234–238, 1987.

8 André Chailloux, Gus Gutoski, and Jamie Sikora. Optimal bounds for semi-honest quantum oblivious transfer. *Chicago Journal of Theoretical Computer Science*, 2016(13):1–17, 2016. URL: http://cjtcs.cs.uchicago.edu/articles/2016/13/contents.html.

9 André Chailloux and Iordanis Kerenidis. Optimal quantum strong coin flipping. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 527–533, 2009. doi:10.1109/FOCS.2009.71.

10 André Chailloux and Iordanis Kerenidis. Optimal bounds for quantum bit commitment. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 354–362, 2011. doi:10.1109/FOCS.2011.42.

11 André Chailloux, Iordanis Kerenidis, and Jamie Sikora. Lower bounds for quantum oblivious transfer. *Quantum Info. Comput.*, 13(1–2):158–177, 2013. doi:10.26421/QIC13.1-2-9.

12 Claude Crépeau. Equivalence between two flavours of oblivious transfers. In *Advances in Cryptology — CRYPTO '87*, pages 350–354, 1988.

13 Claude Crépeau. Quantum oblivious transfer. *Journal of Modern Optics*, 41(12):2445–2454, 1994.

14 Claude Crépeau. Efficient cryptographic protocols based on noisy channels. In *Advances in Cryptology — EUROCRYPT '97*, pages 306–317, 1997. doi:10.1007/3-540-69053-0_21.

15 Ivan B. Damgård, Serge Fehr, Louis Salvail, and Christian Schaffner. Cryptography in the bounded-quantum-storage model. *SIAM Journal on Computing*, 37(6):1865–1890, 2008. doi:10.1137/060651343.

16 Michael J Fischer, Silvio Micali, and Charles Rackoff. A secure protocol for the oblivious transfer. *Journal of Cryptology*, 9(3):191–196, 1996.

17 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *Proceedings of the 19th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

18 Guang Ping He and ZD Wang. Oblivious transfer using quantum entanglement. *Physical Review A—Atomic, Molecular, and Optical Physics*, 73(1):012331, 2006.

19 Yanglin Hu, Yingkai Ouyang, and Marco Tomamichel. Privacy and correctness trade-offs for information-theoretically secure quantum homomorphic encryption. *Quantum*, 7:976, 2023. doi:10.22331/Q-2023-04-13-976.

20 Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 830–842, 2016. doi:10.1145/2976749.2978357.

21 Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the 20th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 20–31, 1988. doi:10.1145/62212.62215.

22 Alexei Kitaev. Quantum coin flipping. Unpublished result. *Talk at the 6th Annual workshop on Quantum Information Processing (QIP 2003)*, 2002.

23 Hoi-Kwong Lo. Insecurity of quantum secure computations. *Phys. Rev. A*, 56:1154–1162, 1997.

24 Carl A Miller. The impossibility of efficient quantum weak coin flipping. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 916–929, 2020. doi:10.1145/3357713.3384276.

25 Carlos Mochon. Quantum weak coin flipping with arbitrarily small bias. arXiv preprint arXiv:0711.4114, 2007.

26 Sarah Anne Osborn. Constant lower bounds on the cryptographic security of quantum two-party computations. Master's thesis, Virginia Tech, 2022.

27 Sarah Anne Osborn and Jamie Sikora. A constant lower bound for any quantum protocol for secure function evaluation. In *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*, volume 232 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:14, 2022. doi:10.4230/LIPICS.TQC.2022.8.

28 James T. Peat and Erika Andersson. Cheating in quantum Rabin oblivious transfer using delayed measurements. arXiv preprint arXiv:2408.12388, 2024. URL: https://arxiv.org/abs/2408.12388.

29 Michael O. Rabin. Digitalized signatures and public-key function as intractable as factorization. *Technical Report, Computer Science, MIT/LCS/TR-212*, 1:100–123, 1979.

30 Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Paper 2005/187, 2005. URL: https://eprint.iacr.org/2005/187.

31 Christian Schaffner. Cryptography in the bounded-quantum-storage model. arXiv preprint arXiv:0709.0289, 2007. arXiv:0709.0289.

32 Christian Schaffner, Barbara Terhal, and Stephanie Wehner. Robust cryptography in the noisy-quantum-storage model. *Quantum Info. Comput.*, 9(11):963–996, 2009. doi:10.26421/QIC9.11-12-4.

33 Lara Stroh, James T. Peat, Mats Kroneberg, Ittoop V. Puthoor, and Erika Andersson. Quantum Rabin oblivious transfer using two pure states. *Phys. Rev. Res.*, 6:043004, 2024.

34 V Vilasini, Christopher Portmann, and Lídia del Rio. Composable security in relativistic quantum cryptography. *New Journal of Physics*, 21(4):043057, 2019.

35 Stephanie Wehner, Christian Schaffner, and Barbara M. Terhal. Cryptography from noisy storage. *Phys. Rev. Lett.*, 100:220502, 2008.

36 Jiawei Wu, Yanglin Hu, Akshay Bansal, and Marco Tomamichel. On the composable security of weak coin flipping. *Quantum*, 9:1780, 2025. doi:10.22331/Q-2025-06-24-1780.

37 Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

A Security proof of Protocol 3

We now provide a proof of Lemma 8. We reproduce the protocol below, for convenience.

- Alice chooses a bit $a \in \{0,1\}$ uniformly at random and creates the two-qutrit state

$$|\phi_a\rangle = \frac{1}{\sqrt{2}}|aa\rangle + \frac{1}{\sqrt{2}}|22\rangle \in \mathcal{A}\otimes\mathcal{B}.$$

- Alice sends the qutrit $\mathcal{B}$ to Bob.
- Bob chooses data bit $y \in \{0,1\}$ and $z \in \{0,1\}$ uniformly at random.
- Bob chooses a permutation bit $p \in \{0,1\}$ uniformly at random and assigns $(x_0, x_1) = (y, z)$ if $p = 0$, or $(x_0, x_1) = (z, y)$, otherwise.
- Bob applies the unitary

$$U_{x_0x_1} = \begin{bmatrix} (-1)^{x_0} & 0 & 0 \\ 0 & (-1)^{x_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  and sends back $\mathcal{B}$ to Alice.
- Alice determines the value of x_a using the two-outcome measurement:

$$\{\Pi_0 \coloneqq |\phi_a\rangle\langle\phi_a|_{\mathcal{A}\otimes\mathcal{B}}, \Pi_1 \coloneqq \mathbb{1}_{\mathcal{A}\otimes\mathcal{B}} - |\phi_a\rangle\langle\phi_a|\}.$$

- Bob sends p to Alice.
- If $a = p$, Alice asserts the receipt of $y = x_a$, else she asserts NULL.

▶ **Lemma 14.** *The optimal success probability (P_A^*) with which dishonest Alice can cheat in Protocol 3 is* $\cos^2(\pi/8)$.

Proof. We have already argued in the main text that the most general strategy for Alice has her creating the state

$$\alpha\,|0\rangle + \beta\,|1\rangle + \gamma\,|2\rangle\,, \tag{16}$$

where α, β, γ are nonnegative and satisfy $\alpha^2 + \beta^2 + \gamma^2 = 1$, and sending this qutrit to Bob. After Bob's unitary, Alice now has the state

$$|\psi_{x_0,x_1}\rangle = \alpha(-1)^{x_0}\,|0\rangle + \beta(-1)^{x_1}\,|1\rangle + \gamma\,|2\rangle\,, \tag{17}$$

which encodes his two bits. Since Alice wants to learn y, she should wait until p is revealed which tells here whether she wants to learn x_0 or x_1 (that is, which bit of (x_0, x_1) is really y). If $p = 0$, that is, Alice wants to learn x_0, her optimal success probability is determined by the Holevo-Helstrom theorem as

$$\frac{1}{2} + \frac{1}{4}\|\rho_0(0) - \rho_1(0)\|_1, \tag{18}$$

where the norm is the trace norm and $\rho_0(0) = \frac{1}{2}\,|\psi_{00}\rangle\,\langle\psi_{00}| + \frac{1}{2}\,|\psi_{01}\rangle\,\langle\psi_{01}|$ is the encoding with $x_0 = 0$ and $\rho_1(0) = \frac{1}{2}\,|\psi_{10}\rangle\,\langle\psi_{10}| + \frac{1}{2}\,|\psi_{11}\rangle\,\langle\psi_{11}|$ is the encoding with $x_0 = 1$. A quick calculation shows that

$$\rho_0(0) - \rho_1(0) = \begin{pmatrix} 0 & 0 & 2\alpha\gamma \\ 0 & 0 & 0 \\ 2\alpha\gamma & 0 & 0 \end{pmatrix} \tag{19}$$

which has at most two nonzero eigenvalues, $\pm 2\alpha\gamma$. Therefore, Alice can cheat with maximum probability

$$\frac{1}{2} + \alpha\gamma \tag{20}$$

in this case.

If $p = 1$, that is, Alice wants to learn x_1, her optimal success probability is again determined by the Holevo-Helstrom theorem as

$$\frac{1}{2} + \frac{1}{4}\|\rho_0(1) - \rho_1(1)\|_1, \tag{21}$$

where $\rho_0(1) = \frac{1}{2}\,|\psi_{00}\rangle\,\langle\psi_{00}| + \frac{1}{2}\,|\psi_{10}\rangle\,\langle\psi_{10}|$ is the encoding with $x_1 = 0$ and $\rho_1(1) = \frac{1}{2}\,|\psi_{01}\rangle\,\langle\psi_{01}| + \frac{1}{2}\,|\psi_{11}\rangle\,\langle\psi_{11}|$ is the encoding with $x_1 = 1$. A quick calculation shows that

$$\rho_0(1) - \rho_1(1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 2\beta\gamma \\ 0 & 2\beta\gamma & 0 \end{pmatrix} \tag{22}$$

which has at most two nonzero eigenvalues, $\pm 2\beta\gamma$. Therefore, Alice can cheat with maximum probability

$$\frac{1}{2} + \beta\gamma \tag{23}$$

in this case.

Since the choice of p is uniformly random, Alice's overall cheating probability is given as

$$\frac{1}{2} + \frac{1}{2}\beta\gamma + \frac{1}{2}\alpha\gamma. \tag{24}$$

A simple optimization over the parameters $\alpha, \beta, \gamma \geq 0$ satisfying $\alpha^2 + \beta^2 + \gamma^2 = 1$ yeilds that $\alpha = \beta = 1/2$ and $\gamma = 1/\sqrt{2}$ is the optimal solution from which we can calculate the optimal cheating probability as $\frac{1}{2} + \frac{1}{2\sqrt{2}} = \cos^2(\pi/8)$, as desired. ◀

▶ **Lemma 15.** *The optimal success probability (P_B^*) with which dishonest Bob can cheat in Protocol 3 is* $3/4$.

Proof. It is necessary and sufficient for dishonest Bob in Protocol 3 to simply learn Alice's bit a as Bob can then set his choice for p accordingly, thereby successfully guessing Alice's assertion of receiving y or NULL. Bob can maximally distinguish the two states $\mathrm{Tr}_{\mathcal{A}}(|\phi_0\rangle\langle\phi_0|)$ and $\mathrm{Tr}_{\mathcal{A}}(|\phi_1\rangle\langle\phi_1|)$ in the Holevo-Helstrom basis resulting in $P_B^* = 3/4$. ◀

Parameterized Verification of Timed Networks with Clock Invariants

Étienne André
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France
Institut Universitaire de France (IUF), Paris, France

Swen Jacobs
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Shyam Lal Karra
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ocan Sankur
Université de Rennes, CNRS, Inria, Rennes, France

Abstract

We consider parameterized verification problems for networks of timed automata (TAs) based on different communication primitives. To this end, we first consider disjunctive timed networks (DTNs), i.e., networks of TAs that communicate via location guards that enable a transition only if there is another process in a certain location. We solve for the first time the case with unrestricted clock invariants, and establish that the parameterized model checking problem (PMCP) over finite local traces can be reduced to the corresponding model checking problem on a single TA. Moreover, we prove that the PMCP for networks that communicate via lossy broadcast can be reduced to the PMCP for DTNs. Finally, we show that for networks with k-wise synchronization, and therefore also for timed Petri nets, location reachability can be reduced to location reachability in DTNs. As a consequence we can answer positively the open problem from Abdulla et al. (2018) whether the universal safety problem for timed Petri nets with multiple clocks is decidable.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Networks of Timed Automata, Parameterized Verification, Timed Petri Nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2408.05190` [13]

Funding *Étienne André*: Partially supported by ANR BisoUS (ANR-22-CE48-0012).
Shyam Lal Karra: Carried out this work as a member of the Saarbrücken Graduate School of Computer Science.

1 Introduction

Formally reasoning about concurrent systems is difficult, in particular if correctness guarantees should hold regardless of the number of interacting processes – a problem also known as *parameterized verification* [3, 7], since the number of processes is considered a parameter of the system. Parameterized verification is undecidable in general [14] and even in very restricted settings, e.g., for safety properties of finite-state processes with rather weak communication primitives, such as token-passing or transition guards [32, 23]. A long line of research has identified classes of systems and properties for which parameterized verification is decidable [23, 28, 24, 25, 22, 18, 9, 26], usually with finite-state processes.

Timed automata (TAs) [8] provide a computational model that combines real-time constraints with concurrency, and are therefore an expressive and widely used formalism to model real-time systems. However, TAs are usually used to model a constant and *fixed* number of system components. When the number n of components is very large or unknown,


© Étienne André, Swen Jacobs, Shyam Lal Karra, and Ocan Sankur;
licensed under Creative Commons License CC-BY 4.0

considering their static combination becomes highly impractical, or even impossible if n is unbounded. However, there are several lines of research studying networks with a parametric number of timed components (see e.g., [6, 17, 4, 11, 1, 10]).

One of these lines considers different variants of timed Petri nets (we here consider the version defined in [2]), and networks of timed automata with k-wise synchronization [6, 5], a closely related model. Due to the expressiveness of the synchronization primitive, results for these models are often negative or limited to severely restricted cases. For example, in networks of timed automata with a controller process and multiple clocks per process, location reachability is undecidable (even in the absence of clock invariants that could force a process to leave a location) [5]. The problem is decidable with a single clock per process and without clock invariants [6]. Decidability remains open for location reachability in networks *without* a controller process and with multiple clocks (with or without clock invariants), which is equivalent to the universal safety problem for timed Petri nets that is mentioned as an open problem in [2].

Another model that has received attention recently and is very important for the work we present is that of *Disjunctive Timed Networks* (DTNs) [31, 12]. It combines the expressive formalism of TAs with the relatively weak communication primitive of disjunctive guards [23]: transitions can be guarded with a location (called "guard location"), and such a transition can only be taken by a TA in the network if another process is in that location upon firing. Consider the example in Figure 1 which illustrates a process's behavior within an asynchronous communication system, where tasks can be dynamically posted and data is read through shared input channels. The transition from init to reading is guarded by location post: for a process to take this transition, at least one other process must be in post. Parameterized model checking of DTNs was first studied in [31], who considered local trace

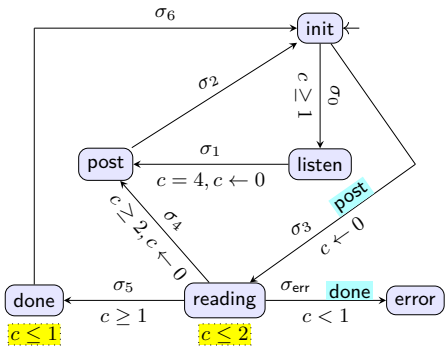


Figure 1 Asynchronous data read example

properties in the temporal logic MITL, and showed that the problem can be solved with a cutoff, i.e., a number of processes that is sufficient to determine satisfaction in networks of any size. However, their result is restricted to the case when guard locations do not have clock invariants. This restriction is crucial to their proof, and they furthermore showed that statically computable cutoffs do not exist for the case when TAs can have clock invariants on all locations.

However, the non-existence of cutoffs does not imply that the problem is undecidable. In [12], the authors improved the aforementioned results by avoiding the construction of a cutoff system and instead using a modified zone graph algorithm. Moreover, they gave sufficient conditions on the TAs to make the problem decidable even in the presence of clock invariants on guard locations. However, these conditions are semantic, and it is not obvious

how to build models that satisfy them; for instance, our motivating example in Figure 1 does not satisfy them. The decidability of the case without restrictions on clock invariants thus remained open.

In this paper, we show that properties of finite local traces (and therefore also location reachability) are decidable for DTNs without restrictions on clock invariants. Moreover, we show that checking local trace properties of systems with lossy broadcast communication [22, 11] or with k-wise synchronization can be reduced to checking local trace properties of DTNs. Note that our simulation of these systems by DTNs crucially relies on the power of clock invariants, and would not be possible in the previous restricted variants of DTNs.

To see why checking local trace properties of DTNs with invariants is technically difficult, consider first the easy case from [31], where guard locations cannot have invariants. In this case, it is enough to determine for every guard location q the minimal time δ at which it can be reached: since a process cannot be forced to leave, q can be occupied at any time in $[\delta, \infty)$, and transitions guarded by q can be assumed to be enabled at any time later than δ. This is already the underlying insight of [31], and in [12] it is embedded into a technique that replaces location guards with clock guards $t \geq \delta$, where t is a clock denoting the time elapsed since the beginning. In contrast, if guard locations can have invariants, a process in q can be forced to leave after some time. Therefore, the set of global times where q can be occupied is an arbitrary set of timepoints, and it is not obvious how it can be finitely represented.

Detailed Example. We introduce an example that motivates the importance of clock invariants in modeling concurrent timed systems, and will be used as a running example. It is inspired by the verification of asynchronous programs [27]. In this setting, processes can be "posted" at runtime to solve a task, and will terminate upon completing the task. Our example in Figure 1 features one clock c per process; symbols σ_i and σ_{err} are transition labels. An unbounded number of processes start in the initial location init. In the inner loop, a process can move to location listen in order to see whether an input channel carries data. Once it determines that this is the case, it moves to post, thereby giving the command to post a process that reads the data, and then can return to init. In the outer loop, if another process gives the command to read data, i.e., is in post, then another process can accept that command and move to reading. After some time, the process will either determine that all the data has been read and move to done, or it will timeout and move to post to ask another process to carry on reading. However, this scheme may run into an error if there are processes in done and reading at the same time, modeled by a transition from reading to error that can only be taken if done is occupied.

While this example is relatively simple, checking reachability of location error (in a network with arbitrarily many processes) is not supported by any existing technique. This is because clock invariants on guard locations are not supported at all by [31], and are supported only in special cases (that do not include this example) by [12]. Also other results that could simulate DTNs do not support clock invariants at all [6, 4].

Moreover, note that clock invariants may be essential for correctness of such systems: in a system A^3, consisting of three copies of the automaton in Figure 1, location error is reachable; a computation that reaches error is given in Figure 2. However, if we add a clock invariant $c \leq 0$ to location post (forcing processes that enter post to immediately leave it again), it becomes unreachable[1].

[1] To see this, consider the intervals of global time in which the different locations can be occupied: first observe that post in the inner loop can now only be occupied in intervals $[4k, 4k]$ (for $k \in \mathbb{N}$), and therefore processes can only move into reading at these times. From there, they might move into post after two time units, so overall post can be occupied in intervals $[2k, 2k]$ for $k \geq 2$, and reading in any

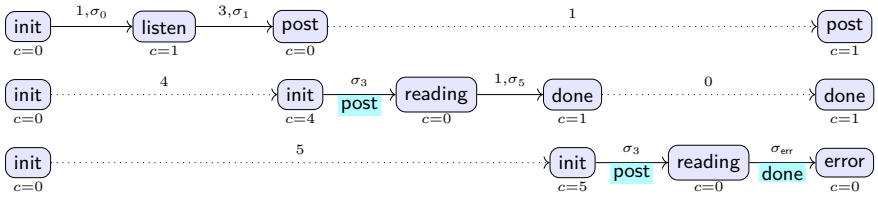


Figure 2 Example of a computation in A^3 for A as depicted by Figure 1.

Contributions. We present new decidability results for parameterized verification problems with respect to three different system models as outlined below.

- **DTNs (Section 3).** For DTNs, we show that, surprisingly, and despite the absence of cutoffs [31], the parameterized model checking problem for finite local traces is decidable in the general case, without any restriction on clock invariants. Our technique circumvents the non-existence of cutoffs by constructing a modified region automaton, a well-known data structure in timed automata literature, such that communication via disjunctive guards is directly taken into account. In particular, we focus on analyzing the traces of a single (or a finite number of) process(es) in a network of arbitrary size.

  While our algorithm uses some techniques from [12], there are fundamental differences: in particular, we introduce a novel abstraction of global time into a finite number of "slots", which are elementary intervals with integer bounds, designed to capture the information necessary for disjunctive guard communication. When a transition with a location guard is to be taken at a given slot, we check whether the given guard location appears in the same slot. It turns out that such an abstract treatment of the global time is sound: we prove that in this case, one can find a computation that enables the location guard at *any* real time instant inside the given slot. Thus, the infinite set of points at which a location guard is enabled is a computable union of intervals; and we rely on this property to build a finite-state abstraction to solve our problem.

- **Lossy Broadcast Timed Networks (Section 4).** We investigate the relation between communication with disjunctive guards and with *lossy broadcast* [22, 11]. For finite-state processes, it is known that lossy broadcast can simulate disjunctive guards wrt. reachability [15], but the other direction is unknown.[2] As our second contribution, we establish the decidability of the parameterized model checking problem for local trace properties in timed lossy broadcast networks. This result is obtained by proving that communication by lossy broadcast is equivalent to communication by disjunctive guards for networks of timed automata *with* clock invariants for local trace properties.

- **Synchronizing Timed Networks and Timed Petri Nets (Section 5).** Finally, we show that the location reachability problem for controllerless multi-clock timed networks with k-wise synchronization can be reduced to the location reachability problem for DTNs with clock invariants.

---

interval $[2k, 2(k+1)]$. Since clock c is always reset upon entering reading, done can only be occupied in intervals $[2k+1, 2k+1]$ for $k \geq 2$, whereas for a process in reading the clock constraint on the transition to error can only be satisfied in intervals $[2k, 2k+1)$. Therefore, error is not reachable with the additional clock invariant on post.

[2] [15] considers IO nets which are equivalent to systems with disjunctive guards. It gives a negative result for a specific simulation relation, but does not prove that simulation is impossible in general.

As a consequence, it follows that the universal safety problem for timed Petri nets with multiple clocks, stated as an open problem in [2], is also decidable.

The proofs of the last two points above involve constructions that require clock invariants on guard locations. This is why clock invariants are crucial in our formalism, which is a nontrivial extension of [12]. Note that in both cases we get decidability even for variants of the respective system models with clock invariants, not considered in [2, 11].

For all of the above systems, location reachability can be decided in EXPSPACE.

Due to space constraints, some details of the definitions and full proofs of some of our results are deferred to the appendix of the extended version [13].

2 Preliminaries

Let $\mathcal{C}$ be a set of clock variables, also called *clocks*. A *clock valuation* is a mapping $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$. For a valuation v and a clock c, we denote the fractional and integral parts of $v(c)$ by $\mathsf{frac}(v(c))$ and $\lfloor v(c) \rfloor$ respectively. We denote by $\mathbf{0}$ the clock valuation that assigns 0 to every clock, and by $v + \delta$ for $\delta \in \mathbb{R}_{\geq 0}$ the valuation s.t. $(v + \delta)(c) = v(c) + \delta$ for all $c \in \mathcal{C}$. Given a subset $\mathcal{C}_r \subseteq \mathcal{C}$ of clocks and a valuation v, $v[\mathcal{C}_r \leftarrow 0]$ denotes the valuation v' such that $v'(c) = 0$ if $c \in \mathcal{C}_r$ and $v'(c) = v(c)$ otherwise. We call *clock constraints* $\Psi(\mathcal{C})$ the terms of the following grammar: $\psi ::= \top \mid \psi \wedge \psi \mid c \sim d \mid c \sim c' + d$ with $d \in \mathbb{N}$, $c, c' \in \mathcal{C}$ and $\sim \in \{<, \leq, =, \geq, >\}$.

A clock valuation v *satisfies* a clock constraint ψ, denoted by $v \models \psi$, if ψ evaluates to $\top$ after replacing every $c \in \mathcal{C}$ with its value $v(c)$.

▶ **Definition 1.** *A* timed automaton (TA) *A is a tuple $(Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$ where Q is a finite set of locations with* initial location *$\hat{q}$, $\mathcal{C}$ is a finite set of clocks, Σ is a finite alphabet that contains a subset Σ^- of special symbols, including a distinguished symbol $\epsilon \in \Sigma^-$, $\mathcal{T} \subseteq Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times Q$ is a transition relation, and $Inv : Q \to \Psi(\mathcal{C})$ assigns to every location q a* clock invariant *$Inv(q)$.*

TAs were introduced in [8] and clock invariants, also simply called invariants, in [29]. We assume w.l.o.g. that invariants only contain upper bounds on clocks (as lower bounds can be moved into the guards of incoming transitions). Σ^- will be used to label silent transitions and unless explicitly specified otherwise (in Sections 4 and 5), we assume that $\Sigma^- = \{\epsilon\}$.

▶ **Example 2.** If we ignore the location guards post (from init to reading) and done (from reading to error), then the automaton in Figure 1 is a classical TA with one clock c. For example, the invariant of done is $c \leq 1$ and the transition from init to reading resets clock c.

A *configuration* of a TA A is a pair (q, v), where $q \in Q$ and $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$ is a clock valuation. A *delay transition* is of the form $(q, v) \xrightarrow{\delta} (q, v + \delta)$ for some delay $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \models Inv(q)$. A *discrete transition* is of the form $(q, v) \xrightarrow{\sigma} (q', v')$, where $\tau = (q, g, \mathcal{C}_r, \sigma, q') \in \mathcal{T}$, $v \models g$, $v' = v[\mathcal{C}_r \leftarrow 0]$ and $v' \models Inv(q')$. A transition $(q, v) \xrightarrow{\epsilon} (q', v')$ is called an *ϵ-transition*. We write $(q, v) \xrightarrow{\delta,\sigma} (q', v')$ if there is a delay transition $(q, v) \xrightarrow{\delta} (q, v + \delta)$ followed by a discrete transition $(q, v + \delta) \xrightarrow{\sigma} (q', v')$.

A *timed path* of A is a finite sequence of transitions $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$. For a timed path $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$, let $\delta(\rho) = \sum_{0 \leq i < l} \delta_i$ be the *total time delay* of ρ. The *length* of ρ is $2l$. A configuration (q, v) has a *timelock* if there is $b \in \mathbb{R}_{\geq 0}$ s.t. $\delta(\rho) \leq b$ for every timed path ρ starting in (q, v). We write $(q_0, v_0) \to^* (q_l, v_l)$ if there is a timed path $\rho = (q_0, v_0) \xrightarrow{\delta_0,\sigma_0} \dots \xrightarrow{\delta_{l-1},\sigma_{l-1}} (q_l, v_l)$; ρ is a *computation* if $q_0 = \hat{q}$ and $v_0 = \mathbf{0}$.

The *trace* of the timed path ρ is the sequence of pairs of delays and labels obtained by removing transitions with a label from Σ^- and adding the delays of these to the following transition. The *language* of A, denoted $\mathcal{L}(A)$, is the set of traces of all of its computations.

We now recall guarded timed automata as an extension of timed automata with location guards, that will allow, in a network, to test whether some other process is in a given location in order to pass the guard.

▶ **Definition 3** (Guarded Timed Automaton (GTA)). *A* GTA A *is a tuple* $(Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$*, where* Q *is a finite set of locations with* initial location $\hat{q}$*,* $\mathcal{C}$ *is a finite set of clocks,* Σ *is a finite alphabet that contains a subset* Σ^- *of special symbols, including a distinguished symbol* $\epsilon \in \Sigma^-$*,* $\mathcal{T} \subseteq Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times (Q \cup \{\top\}) \times Q$ *is a transition relation, and* $Inv : Q \to \Psi(\mathcal{C})$ *assigns to every location* q *an* invariant $Inv(q)$*.*

Intuitively, a transition $\tau = (q, g, \mathcal{C}_r, \sigma, \gamma, q') \in \mathcal{T}$ takes the automaton from location q to q'; τ can only be taken if *clock guard* g and *location guard* γ are both satisfied, and it resets all clocks in $\mathcal{C}_r$. Note that satisfaction of location guards is only meaningful in a *network* of TAs (defined below). Intuitively, a location guard γ is satisfied if it is $\top$ or if another automaton in the network currently occupies location γ. We say that γ is *trivial* if $\gamma = \top$. We say location q *has no invariant* if $Inv(q) = \top$.

▶ **Example 4.** In the GTA in Figure 1, the transition from init to reading is guarded by location guard post. The transition from init to listen has a trivial location guard (trivial location guards are not depicted in our figures). Location init has no invariant.

▶ **Definition 5** (Unguarded Timed Automaton). *Given a GTA* A*, we denote by* $\mathsf{UG}(A)$ *the* unguarded *version of* A*, which is the TA obtained from* A *by removing location guards, and adding a fresh clock* t*, called the global clock, that does not appear in the guards or resets. Formally,* $\mathsf{UG}(A) = (Q, \hat{q}, \mathcal{C} \cup \{t\}, \mathcal{T}', Inv)$ *with* $\mathcal{T}' = \big\{(q, g, \mathcal{C}_r, \sigma, q') \mid (q, g, \mathcal{C}_r, \sigma, \gamma, q') \in \mathcal{T}\big\}$*.*

For a GTA A, let A^n denote a *network of guarded timed automata* (NGTA), consisting of n copies of A. Each copy of A in A^n is called a *process*.

A *configuration* $\mathfrak{c}$ of an NGTA A^n is a tuple $\big((q_1, v_1), \ldots, (q_n, v_n)\big)$, where every (q_i, v_i) is a configuration of A. The semantics of A^n can be defined as a *timed transition system* $(\mathfrak{C}, \hat{\mathfrak{c}}, T)$, where $\mathfrak{C}$ denotes the set of all configurations of A^n, $\hat{\mathfrak{c}}$ is the unique initial configuration $(\hat{q}, \mathbf{0})^n$, and the transition relation T is the union of the following delay and discrete transitions:

delay transition $\big((q_1, v_1), \ldots, (q_n, v_n)\big) \xrightarrow{\delta} \big((q_1, v_1 + \delta), \ldots, (q_n, v_n + \delta)\big)$ if $\forall i \in \{1, \ldots, n\}$: $v_i + \delta \models Inv(q_i)$, i.e., we can delay $\delta \in \mathbb{R}_{\geq 0}$ units of time if all clock invariants are satisfied at the end of the delay.

discrete transition $\big((q_1, v_1), \ldots, (q_n, v_n)\big) \xrightarrow{(i,\sigma)} \big((q_1', v_1'), \ldots, (q_n', v_n')\big)$ for some $i \in \{1, \ldots, n\}$ if

1) $(q_i, v_i) \xrightarrow{\sigma} (q_i', v_i')$ is a discrete transition of A with $\tau = (q_i, g, \mathcal{C}_r, \sigma, \gamma, q_i')$,

2) $\gamma = \top$ or $q_j = \gamma$ for some $j \in \{1, \ldots, n\} \setminus \{i\}$, and

3) $q_j' = q_j$ and $v_j' = v_j$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$.

That is, location guards γ are interpreted as disjunctive guards: unless $\gamma = \top$, at least one other process needs to occupy location γ in order for process i to pass this guard.

We write $\mathfrak{c} \xrightarrow{\delta,(i,\sigma)} \mathfrak{c}''$ for a delay transition $\mathfrak{c} \xrightarrow{\delta} \mathfrak{c}'$ followed by a discrete transition $\mathfrak{c}' \xrightarrow{(i,\sigma)} \mathfrak{c}''$. Then, a *timed path* of A^n is a finite sequence $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$.

For a timed path $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$, let $\delta(\pi) = \sum_{0 \le i < l} \delta_i$ be the total time delay of π. The definition of timelocks extends naturally to configurations of NGTAs. A timed path π of A^n is a *computation* if $\mathfrak{c}_0 = \hat{\mathfrak{c}}$. Its *length* is equal to $2l$.

We write $q \in \mathfrak{c}$ if $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ and $q = q_i$ for some $i \in \{1, \ldots, n\}$, and similarly $(q, v) \in \mathfrak{c}$. We say that a location q is *reachable* in A^n if there exists a reachable configuration $\mathfrak{c}$ s.t. $q \in \mathfrak{c}$.

▶ **Example 6.** Consider the NGTA A^3 where A is the GTA shown in Figure 1. A computation π of this network is depicted in Figure 2, in which a process reaches error with $\delta(\pi) = 5$. The computation is $\big((\mathsf{init}, c = 0), (\mathsf{init}, c = 0), (\mathsf{init}, c = 0)\big) \xrightarrow{1,(1,\sigma_0)} \big((\mathsf{listen}, c = 1), (\mathsf{init}, c = 1), (\mathsf{init}, c = 1)\big) \xrightarrow{3,(1,\sigma_1)} \big((\mathsf{post}, c = 0), (\mathsf{init}, c = 4), (\mathsf{init}, c = 4)\big) \xrightarrow{0,(2,\sigma_3)} \big((\mathsf{post}, c = 0), (\mathsf{reading}, c = 0), (\mathsf{init}, c = 4)\big) \xrightarrow{1,(2,\sigma_5)} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{init}, c = 5)\big) \xrightarrow{0,(3,\sigma_3)} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{reading}, c = 0)\big) \xrightarrow{0,(3,\sigma_{\mathsf{err}})} \big((\mathsf{post}, c = 1), (\mathsf{done}, c = 1), (\mathsf{error}, c = 0)\big)$. Therefore, error is reachable in A^3.

The *trace* of the timed path π is a sequence $\mathsf{trace}(\pi) = (\delta'_0, (i'_0, \sigma'_0)) \ldots (\delta'_{l-1}, (i'_{l-1}, \sigma'_{l-1}))$ obtained by removing all discrete transitions (j, σ_j) of π with $\sigma_j \in \Sigma^-$, and adding all delays of these transitions to the following discrete transition, yielding the δ'_j. The *language* of A^n, denoted $\mathcal{L}(A^n)$, is the set of traces of all of its computations.

▶ **Example 7.** For the computation π in Example 6, $\mathsf{trace}(\pi) = \big(1, (1, \sigma_0)\big), \big(3, (1, \sigma_1)\big), \big(0, (2, \sigma_3)\big), \big(1, (2, \sigma_5)\big), \big(0, (3, \sigma_3)\big), \big(0, (3, \sigma_{\mathsf{err}})\big)$.

We will also use *projections* of these global objects onto subsets of the processes. That is, if $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ and $\mathcal{I} = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, then $\mathfrak{c}{\downarrow_{\mathcal{I}}}$ is the tuple $\big((q_{i_1}, v_{i_1}), \ldots, (q_{i_k}, v_{i_k})\big)$, and we extend this notation to computations $\pi{\downarrow_{\mathcal{I}}}$ by keeping only the discrete transitions of $\mathcal{I}$ and by adding the delays of the removed discrete transitions to the delay of the following discrete transition of $\mathcal{I}$.

We introduce a special notation for projecting to a single process and define, for any natural number $1 \le a \le n$, $\pi{\downarrow_a}$ a computation of $\mathsf{UG}(A)$, obtained from $\pi{\downarrow_{\{a\}}}$ by discarding the index a from all transitions; that is, $\pi{\downarrow_a}$ has the form $(q_0, v_0) \xrightarrow{(\delta'_{k_0}, \sigma_{k_0})} \ldots \xrightarrow{(\delta'_{k_m}, \sigma_{k_m})} (q_{m+1}, v_{m+1})$. We also extend this to traces; that is, $\mathsf{trace}(\pi){\downarrow_a} = (\delta'_{k_0}, \sigma_{k_0}) \ldots (\delta'_{k_m}, \sigma_{k_m})$, which is a trace of $\mathsf{UG}(A)$. For a set of traces L, and set $\mathcal{I}$ of processes, we write $L{\downarrow_{\mathcal{I}}} = \{tt{\downarrow_{\mathcal{I}}} \mid tt \in L\}$.

Note that the projection of a computation is not necessarily a computation itself, since location guards may not be satisfied.

▶ **Example 8.** For the computation π in Example 6, $\pi{\downarrow_3} = (\mathsf{init}, c = 0) \xrightarrow{(5,\sigma_3)} (\mathsf{reading}, c = 0) \xrightarrow{(0,\sigma_{\mathsf{err}})} (\mathsf{error}, c = 0)$ and $\mathsf{trace}(\pi){\downarrow_3} = (5, \sigma_3), (0, \sigma_{\mathsf{err}})$.

A *prefix* of a computation $\pi = \mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l-1},(i_{l-1},\sigma_{l-1})} \mathfrak{c}_l$, is a sequence $\mathfrak{c}_0 \xrightarrow{\delta_0,(i_0,\sigma_0)} \cdots \xrightarrow{\delta_{l'},(i_{l'},\sigma_{l'})} \mathfrak{c}_{l'}$ with $l' \le l - 1$. If π is a timed path and $d \in \mathbb{R}_{\ge 0}$, then $\pi^{\le d}$ denotes the maximal prefix of π with $\delta(\pi^{\le d}) \le d$, and similarly for timed paths $\rho^{\le d}$ of a single GTA. For timed paths π_1 of A^{n_1} and π_2 of A^{n_2} with $\delta(\pi_1) = \delta(\pi_2)$, we denote by $\pi_1 \parallel \pi_2$ their *composition* into a timed path of $A^{n_1+n_2}$ whose projection to the first n_1 processes is π_1, and whose projection to the last n_2 processes is π_2.

▶ **Definition 9** (Disjunctive Timed Network)**.** *A given GTA A induces a* disjunctive timed network (DTN) *A^∞, defined as the following family of NGTAs: $A^\infty = \{A^n \mid n \in \mathbb{N}_{>0}\}$ (we follow the terminology and use abbreviations of [31]). We define $\mathcal{L}(A^\infty) = \bigcup_{n \in \mathbb{N}_{>0}} \mathcal{L}(A^n)$ and consider $\mathcal{L}(A^\infty){\downarrow_I} = \bigcup_{n \in \mathbb{N}_{>0}} \mathcal{L}(A^n){\downarrow_I}$.*

2.1 The Parameterized Model Checking Problem

We formalize properties of DTNs as sets of traces that describe the intended behavior of a fixed number of processes running in a system with arbitrarily many processes. That is, a *local property* φ of k processes, also called a *k-indexed property*, is a subset of $(\mathbb{R}_{\geq 0} \times ([1,k] \times \Sigma))^*$. For $k = 1$, for simplicity, we consider it as a subset of $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. We say that A^n *satisfies* a k-indexed local property φ, denoted $A^n \models \varphi$, if $\mathcal{L}(A^n)\downarrow_{[1,k]} \subseteq \varphi$. Note that, due to the symmetry of the system, it does not matter *which* k processes we project $\mathcal{L}(A^n)$ onto, so we always project onto the first k.

Parameterized model checking problem (PMCP):
INPUT: a GTA A and a k-indexed local property φ
PROBLEM: Decide whether $A^n \models \varphi$ holds $\forall n \geq k$.

Local trace properties allow to specify for instance any local safety property of a single process (with $I = [1,1]$), as well as mutual exclusion properties (with $I = [1,2]$) and variants of such properties for larger I.

PMCP can be solved by checking whether $\mathcal{L}(A^\infty)\downarrow_{[1,k]} \subseteq \varphi$. Our solution consists in building a TA that recognizes $\mathcal{L}(A^\infty)\downarrow_{[1,k]}$. Note that language inclusion is undecidable on TAs [8], but many interesting problems are decidable. These include MITL model checking [21] and simpler problems such as reachability: given symbol $\sigma_0 \in \Sigma$, the *reachability PMCP* is the PMCP where φ is the set of traces that contain an occurrence of σ_0. Reachability of a *location* of A can be solved by PMCP by choosing appropriate transition labels.

▶ **Example 10.** In the example of Figure 1, a natural local property we are interested in is the reachability of the label σ_{err}. Formally, the local property for process 1 can be written as a 1-indexed property: $(\mathbb{R}_{\geq 0} \times ([1,1] \times \Sigma))^* \cdot \{(d, (1, \sigma_{\mathsf{err}})) \mid d \in \mathbb{R}_{\geq 0}\} \cdot (\mathbb{R}_{\geq 0} \times ([1,1] \times \Sigma))^*$.

3 Model Checking DTNs

3.1 Definitions

We recall here the standard notions of regions and region automata, and introduce the *slots* of regions which refer to the intervals of possible valuations of a global clock.

Regions. Given A, for all $c \in \mathcal{C}$, let $M(c)$ denote the maximal bound that c is compared to: $M(c) = \max\{d \in \mathbb{Z} \mid$ "$c \sim d$", "$c - c' \sim d$", "$c' - c \sim d$" appears in a guard or invariant of $A\}$(we set $M(c) = 0$ if this set is empty). M is called the *maximal bound function for* A. Define $M_{\max} = \max\{M(c) \mid c \in \mathcal{C}\}$. We say that two valuations v and v' are equivalent w.r.t. M, denoted by $v \simeq_M v'$, if the following conditions hold for any clocks c, c' [19, 20]:

1. either $\lfloor v(c) \rfloor = \lfloor v'(c) \rfloor$ or $v(c) > M(c)$ and $v'(c) > M(c)$;
2. if $v(c), v'(c) \leq M(c)$ then $\mathsf{frac}(v(c)) = 0 \iff \mathsf{frac}(v'(c)) = 0$;
3. if $v(c) \leq M(c), v(c') \leq M(c')$ then $\mathsf{frac}(v(c)) \leq \mathsf{frac}(v(c')) \iff \mathsf{frac}(v'(c)) \leq \mathsf{frac}(v'(c'))$;
4. for any interval I among $(-\infty, -M(c')), [-M(c'), -M(c')], (-M(c'), -M(c') + 1), \ldots, [M(c), M(c)], (M(c), \infty)$, we have $v(c) - v(c') \in I \iff v'(c) - v'(c') \in I$.

An *M-region* is an equivalence class of valuations induced by $\simeq_M$. We denote by $[v]_M$ the region to which v belongs. We omit M when it is clear from context.

For an M-region r, if a valuation $v \in r$ satisfies a clock guard g, then every valuation in r satisfies g. We write $r \models g$ to mean that every valuation in r satisfies g.

Given an M-region r and a clock c, let $r{\downarrow_c}$ denote the projection of the valuations of r to c, i.e., $r{\downarrow_c} = \{v(c) \mid v \in r\}$. Given a valuation v and a clock $c \in \mathcal{C}$, let $v{\downarrow_{-c}}$ denote *the projection of v to the clocks other than c*, i.e., $v{\downarrow_{-c}} \colon \mathcal{C} \setminus \{c\} \to \mathbb{R}_{\geq 0}$ is defined by $v{\downarrow_{-c}}(c') = v(c')$ for all $c' \in \mathcal{C} \setminus \{c\}$. By extension, given a region r and a clock c, let $r{\downarrow_{-c}} = \{v{\downarrow_{-c}} \mid v \in r\}$.

Region Automaton. The *region automaton* of a TA A is a finite automaton with alphabet $\Sigma \cup \{\mathsf{delay}\}$, denoted by $\mathcal{R}_M(A)$, defined as follows.

The *region states* are pairs (q, r) where $q \in Q$ and r is an M-region. The *initial region state* is $(\hat{q}, \hat{r})$ where $\hat{q}$ is the initial location of A and $\hat{r}$ is the singleton region containing $\mathbf{0}$.

There is a transition $(q, r) \xrightarrow{\mathsf{delay}} (q, r')$ in $\mathcal{R}_M(A)$ iff there is a transition $(q, v) \xrightarrow{\delta} (q, v')$ in A for some $\delta \in \mathbb{R}_{\geq 0}, v \in r$ and $v' \in r'$. We say that r' is a *time successor* of r. Note that we can have $r' = r$. Furthermore, (q, r') is the *immediate time successor* of (q, r) if $(q, r) \xrightarrow{\mathsf{delay}} (q, r')$, $r' \neq r$, and whenever $(q, r) \xrightarrow{\mathsf{delay}} (q, r'')$, we have $(q, r') \xrightarrow{\mathsf{delay}} (q, r'')$.

There is a transition $(q, r) \xrightarrow{\sigma} (q', r')$ in $\mathcal{R}_M(A)$ iff there is a transition $(q, v) \xrightarrow{\sigma} (q', v')$ with label σ in A for some $v \in r$ and $v' \in r'$. We write $(q, r) \to (q', r')$ if either $(q, r) \xrightarrow{\mathsf{delay}} (q', r')$ or $(q, r) \xrightarrow{\sigma} (q', r')$ for some $\sigma \in \Sigma$.

A *path* in $\mathcal{R}_M(A)$ is a finite sequence of transitions $\rho_r = (q_0, r_0) \xrightarrow{\sigma_0} \dots \xrightarrow{\sigma_{n-1}} (q_n, r_n)$ for some $n \geq 0$ where $\sigma_i \in \Sigma \cup \{\mathsf{delay}\}$. A path of $\mathcal{R}_M(A)$ is a *computation* if it starts from the initial region state.

It is known that $\mathcal{R}_M(A)$ captures the *untimed* traces of A, i.e., the projection of the traces of A to Σ [8].

Slots. Now, we can introduce slots. We will show later that slots are a sufficiently precise abstraction of time for DTNs. In this paragraph, we assume that TAs have a distinguished *global clock* t which is never reset and does not appear in clock guards. We will thus consider a clock set $\mathcal{C} \cup \{t\}$ (making t appear explicitly for clarity).

Let N_A denote the number of pairs (q, r) where $q \in Q$ and r is an M-region (thus a region on the clock set $\mathcal{C}$ without t). Recall that N_A is exponential in $|\mathcal{C}|$ [8, 20]. Let us consider a bound function $M^{\nearrow} : \mathcal{C} \cup \{t\} \to \mathbb{N}$ for A such that for $c \in \mathcal{C} \setminus \{t\}$, $M^{\nearrow}(c) = M(c)$, and $M^{\nearrow}(t) = 2^{N_A+1}$. Throughout the paper, the bound functions will be denoted by $M^{\nearrow}(\cdot)$ whenever the clock set contains the distinguished global clock t, and $M(\cdot)$ otherwise. The former will be referred to as $M^{\nearrow}$-regions, and the latter as M-regions.

We define the *slot of an $M^{\nearrow}$-region r* as $\mathsf{slot}(r) = r{\downarrow_t}$. It is known that for any region r (with any bound function) and clock c, $r{\downarrow_c}$ is an interval [30]. Moreover, if $v(c)$ for every $v \in r$ is below the maximal constant $M^{\nearrow}(c)$, then $r{\downarrow_c}$ is either a singleton interval of the form $[k, k]$, or an open interval of the form $(k, k+1)$ for some $k \in \mathbb{N}$.

For a slot s, let us define $\mathsf{next}(s)$ as follows.

1. if $s = (k, k+1)$ for some $k \in \mathbb{N}$, then $\mathsf{next}(s) = [k+1, k+1]$;
2. if $s = [k, k]$ and $k < M^{\nearrow}(t)$, then $\mathsf{next}(s) = (k, k+1)$;
3. if $s = [M^{\nearrow}(t), M^{\nearrow}(t)]$, then $\mathsf{next}(s) = (M^{\nearrow}(t), \infty)$.
4. if $s = (M^{\nearrow}(t), \infty)$ then $\mathsf{next}(s) = s$.

We define the *slot of a valuation v* on $\mathcal{C} \cup \{t\}$ as $\mathsf{slot}(r)$ where r is the (unique) $M^{\nearrow}$-region s.t. $v \in r$. Slots, seen as intervals, can be bounded or unbounded.

▶ **Example 11.** Consider the clock set $\{x, y, t\}$ and the region r defined by $\lfloor x \rfloor = \lfloor y \rfloor = 1$, $\lfloor t \rfloor = 2$, $0 < \mathsf{frac}(x) < \mathsf{frac}(y) < \mathsf{frac}(t) < 1$ (with $M^{\nearrow}(\cdot) = 4$ for all clocks). Then, $\mathsf{slot}(r) = (2, 3)$.

As a second example, assume $M^{\nearrow}(x) = 2$, $M^{\nearrow}(y) = 3$ and $M^{\nearrow}(t)$ is, say, 2048. Consider the region r' defined by $x > 2 \wedge \lfloor y \rfloor = 1 \wedge 0 < \mathsf{frac}(y) < 1 \wedge \lfloor t \rfloor = 2048 \wedge \mathsf{frac}(t) = 0$. Then, $\mathsf{slot}(r) = [2048, 2048]$. In addition, $\mathsf{next}(\mathsf{slot}(r)) = (2048, \infty)$.

We now introduce the *shifting* operation which consists of increasing the global clock value, without changing the values of other clocks.

▶ **Lemma 12.** *Given any $M^{\nearrow}$-region r and $k \in \mathbb{Z}$ such that $\sup(\mathsf{slot}(r)) + k \leq M^{\nearrow}(t)$, and $\inf(\mathsf{slot}(r)) + k \geq 0$, there exists a $M^{\nearrow}$-region r' which satisfies $\mathsf{slot}(r') = \mathsf{slot}(r) + k$ and $r'{\downarrow_{-t}} = r{\downarrow_{-t}}$, and r' can be computed in polynomial time in the number of clocks.*

The region r' in Lemma 12 will be denoted by $r_{\mathsf{slot}+k}$. We say that it is obtained by *shifting the slot by k in r*. We extend this notation to sets of regions and sets of region states, that is, $W_{\mathsf{slot}+k} = \{(q, r_{\mathsf{slot}+k}) \mid (q, r) \in W\}$ where W is a set of region states. For a set of region states W, we define $W{\downarrow_{-t}} = \{(q, r{\downarrow_{-t}}) \mid (q, r) \in W\}$.

▶ **Example 13.** Consider the clock set $\{x, y, t\}$ and the region r defined in Example 11 satisfying $\lfloor x \rfloor = \lfloor y \rfloor = 1$, $\lfloor t \rfloor = 2$, $0 < \mathsf{frac}(x) < \mathsf{frac}(y) < \mathsf{frac}(t) < 1$ (with $M^{\nearrow}_{max} = 4$). Then, $\mathsf{slot}(r) = (2, 3)$, and $r_{\mathsf{slot}+1}$ is defined by the same constraints as above except that $\lfloor t \rfloor = 3$, and $\mathsf{slot}(r_{\mathsf{slot}+1}) = (3, 4)$.

▶ Remark 14. Recall that given a bound function, the number of regions is $O(|\mathcal{C}|!2^{|\mathcal{C}|}{M_{\max}}^{|\mathcal{C}|})$ since regions determine an order of the fractional values of clocks, the subsets of clocks that have integer values, and an integral value for each clock [19]. The number of $M^{\nearrow}$-regions is $O\left(|\mathcal{C}|!2^{|\mathcal{C}|}(M^{\nearrow}(t))^{|\mathcal{C}|}\right)$, which is doubly exponential in $|\mathcal{C}|$ since $M^{\nearrow}(t)$ is.

Crucial to our paper, however, is that the set of *projections* $r{\downarrow_{-t}}$ of the set of $M^{\nearrow}$-regions r has size exponential only. This can be seen as follows: our definition of regions from [19] uses a distinct maximum bound function for each clock. Thus, when constraints on t are eliminated, there only remain constraints on clocks $c \in \mathcal{C} \setminus \{t\}$, with maximal constants $M(c)$ as in the original GTA A. We thus fall back to the set of regions of A of size $O(|\mathcal{C}|!2^{|\mathcal{C}|}{M_{\max}}^{|\mathcal{C}|})$.

3.2 Layer-based Algorithm for the DTN Region Automaton

We describe here an algorithm to compute a TA $\mathcal{S}(A)$ that recognizes the language $\mathcal{L}(A^{\infty}){\downarrow_1}$. We explain at the end of the section how to generalize the algorithm to compute $\mathcal{L}(A^{\infty}){\downarrow_I}$ for an interval $I = [1, a]$ for $a \geq 1$.

▶ **Assumption 1.** *We assume that the given GTA A is* timelock-free*, regardless of location guards. Formally, let A' be obtained from A by removing all transitions with non-trivial location guards. We require that no configuration of A' has a timelock.*

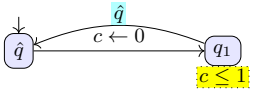


Figure 3 A GTA with timelock due to location guards.

Note that this assumption guarantees that A^n will be timelock-free for all n. Assuming timelock-freeness is not restrictive since a protocol cannot possibly block the physical time: time will elapse regardless of the restrictions of the design. A network with a timelock is thus a design artifact, and just means the model is incomplete. An incomplete model can be completed by adding a sink location to which processes that would cause a timelock can move, and regarding reachability the resulting model is equivalent to the original one.

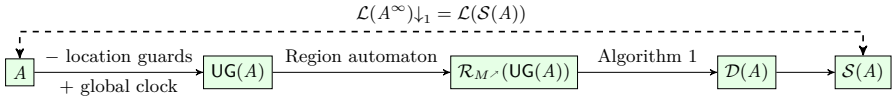


Figure 4 An overview of data structures in the paper.

▶ **Example 15.** The GTA in Figure 3 does not satisfy Assumption 1, since A' (where we remove transitions with non-trivial location guards) has a timelock at $(q_1, c = 1)$.

The following assumption simplifies the proofs:

▶ **Assumption 2.** *Each transition of GTA A is labeled by a unique label different from ϵ.*

Consider a GTA A. Our algorithm builds a TA capturing the language $\mathcal{L}(A^\infty)\downarrow_1$. The construction is based on $M^\nearrow$-region states of $\mathsf{UG}(A)$; however, not all transitions of the region automaton of $\mathsf{UG}(A)$ are to be added since location guards mean that some transitions are not enabled at a given region. Unless otherwise stated, by region states we mean $M^\nearrow$-region states. The steps of the construction are illustrated in Figure 4. From A, we first obtain $\mathsf{UG}(A)$, and build the region automaton for $\mathsf{UG}(A)$, denoted by $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$. Then Algorithm 1 builds the so-called *DTN region automaton* $\mathcal{D}(A)$ which is a finite automaton. Finally we construct $\mathcal{S}(A)$ which we refer to as the *summary timed automaton*, a timed automaton derived from $\mathcal{D}(A)$ by adding clocks and clock guards to $\mathcal{D}(A)$. Our main result is that $\mathcal{S}(A)$ recognizes the language $\mathcal{L}(A^\infty)\downarrow_1$.

Intuitively, Algorithm 1 computes region states reachable by a single process within the context of a network of arbitrary size. These region states are partitioned according to their slots. More precisely, Algorithm 1 computes (lines 3-4) the sequence $(W_i, E_i)_{i\geq 0}$, where W_i is a set of $M^\nearrow$-region states of $\mathsf{UG}(A)$ having the same slot (written $\mathsf{slot}(W_i)$), and E_i is a set of transitions from region states of W_i to either W_i or W_{i+1}. These transitions include ϵ-transitions which correspond to delay transitions: if the slot does not change during the delay transition, then the transition goes to a region-state which is also in W_i; otherwise, it leaves to the next slot and the successor is in W_{i+1}. During discrete transitions from W_i, the slot does not change, so the successor region-states are always inside W_i. In order to check if a discrete transition with location guard γ must be considered, the algorithm checks if some region-state (γ, r_γ) was previously added to the *same* layer W_i. This means that some (other) process can be at γ somewhere at a global time that belongs to $\mathsf{slot}(W_i)$. This is the nontrivial part of the algorithm: the proof will establish that if a process can be at location γ at *some* time in a given slot s, then it can also be at γ at *any* time within s.

For two sets W_i, W_j of region states of $\mathsf{UG}(A)$, let us define $W_i \approx W_j$ iff W_j can be obtained from W_i by shifting the slot, that is, if there exists $k \in \mathbb{Z}$ such that $(W_i)_{\mathsf{slot}+k} = W_j$. Recall that $(W_i)_{\mathsf{slot}+k} = W_j$ means that both sets contain the same regions when projected to the local clocks $\mathcal{C} \setminus \{t\}$. This definition is of course symmetric.

Algorithm 1 stops (line 5) when $W_{i_0} \approx W_{l_0}$ for some $i_0 < l_0$ with both layers having singleton slots (this requirement could be relaxed but this simplifies the proofs and only increases the number of iterations by a factor of 2).

The algorithm returns the DTN region automaton $\mathcal{D}(A) = (W, (\hat{q}, \hat{r}), \Sigma, E)$, where W is the set of explored region states, and E is the set of transitions that were added; except that transitions leaving W_{l_0-1} are redirected back to W_{i_0} (lines 7–9). Redirecting such transitions means that whenever $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$ has a delay transition from (q, r) to (q, r') with $\mathsf{slot}(r) = \mathsf{slot}(W_{l_0-1})$ and $\mathsf{slot}(r') = \mathsf{slot}(W_{l_0})$, then we actually add a transition from (q, r) to (q, r''), where r'' is obtained from r' by shifting the slot to that of $\mathsf{slot}(W_{i_0})$; this means that $r'\downarrow_{-t} = r''\downarrow_{-t}$, so these define the same clock valuations except with a shifted slot. The property $W_{i_0} \approx W_{l_0}$ ensures that $(q, r'') \in W_{i_0}$.

Algorithm 1 Algorithm to compute DTN region automaton of GTA A.

input : GTA $A = (Q, \hat{q}, \mathcal{C}, \Sigma, \mathcal{T}, Inv)$ and $\mathcal{R}_{M^{\nearrow}}(\mathsf{UG}(A))$
output : The DTN region automaton of A

Initialize $s \leftarrow [0,0]$, $W_0 \leftarrow \{(\hat{q}, \hat{r})\}$, $E_0 \leftarrow \emptyset$, $l \leftarrow -1$
repeat
  $l \leftarrow l + 1$;
  Compute (W_l, E_l) by applying the following rules until a fixed point is reached:
  - Rule 1: For any $(q,r) \in W_l$, let $(q,r) \xrightarrow{\mathsf{delay}} (q,r')$ s.t. $\mathsf{slot}(r') = s$, do
    $W_l \leftarrow W_l \cup \{(q,r')\}$, and $E_l \leftarrow E_l \cup \left\{\big((q,r), \epsilon, (q,r')\big)\right\}$.
  - Rule 2: For any $(q,r) \in W_l$ and $\tau = (q, g, \mathcal{C}_r, \sigma, \gamma, q')$ s.t. $(q,r) \xrightarrow{\sigma} (q',r')$, if $\gamma = \top$, or if there exists $(\gamma, r_\gamma) \in W_l$,
    then do $W_l \leftarrow W_l \cup \{(q',r')\}$, and $E_l \leftarrow E_l \cup \left\{\big((q,r), \sigma, (q',r')\big)\right\}$

  $s \leftarrow \mathsf{next}(s)$;
  $W_{l+1} \leftarrow \{(q,r') \mid (q,r) \in W_l, (q,r) \xrightarrow{\mathsf{delay}} (q,r') \text{ and } \mathsf{slot}(r') = s\}$;
  $E_l \leftarrow E_l \cup \left\{\big((q,r), \epsilon, (q,r')\big) \mid (q,r) \in W_l,\ (q,r) \xrightarrow{\mathsf{delay}} (q,r') \wedge \mathsf{slot}(r') = s\right\}$;
until $\exists i_0 < l : W_l \approx W_{i_0}$, *and* $\mathsf{slot}(W_{i_0})$ *is a singleton*;
$l_0 \leftarrow l$;
$E_{l_0-1} \leftarrow E_{l_0-1} \cap \big(W_{l_0-1} \times \Sigma \times W_{l_0-1}\big) \cup$
$\left\{\big((q,r), \epsilon, (q,r')\big) \,\middle|\, (q,r') \in W_{i_0}, \exists r'', \exists k \in \mathbb{N}, \big((q,r), \epsilon, (q,r'')\big) \in E_{l_0-1} \cap \big(W_{l_0-1} \times \Sigma \times W_{l_0}\big), r'_{\mathsf{slot}+k} = r''\right\}$
$W \leftarrow \cup_{0 \le i \le l_0-1} W_i$, $E \leftarrow \cup_{0 \le i \le l_0-1} E_i$
return $(W, (\hat{q}, \hat{r}), \Sigma, E)$

We write $(q,r) \xRightarrow{\sigma} (q',r')$ iff $\big((q,r), \sigma, (q',r')\big) \in E$. Paths and computations are defined for the DTN region automaton analogously to region automata.

We now show how to construct the summary timed automaton $\mathcal{S}(A)$ (the step from $\mathcal{D}(A)$ to $\mathcal{S}(A)$ in Figure 4). We define $\mathcal{S}(A)$ by extending $\mathcal{D}(A)$ with the clocks of A. Moreover, each transition $((q,r), \epsilon, (q,r'))$ has the guard $r'{\downarrow_{-t}}$ and no reset; and each transition $((q,r), \sigma, (q',r'))$ with $\sigma \neq \epsilon$ has the guard $r{\downarrow_{-t}}$, and resets the clocks that are equal to 0 in r'. The intuition is that $\mathcal{S}(A)$ ensures by construction that any valuation that is to take a discrete transition ($\sigma \neq \epsilon$) at location (q,r) belongs to r. Notice that we omit invariants here. Because transitions are derived from those of $\mathcal{R}_{M^{\nearrow}}(\mathsf{UG}(A))$, the only additional behavior we can have in $\mathcal{S}(A)$ due to the absence of invariants is a computation delaying in a location (q,r) and reaching outside of r (without taking an ϵ-transition), while no discrete transitions can be taken afterwards. Because traces end with a discrete transition, this does not add any trace not possible in $\mathcal{L}{\downarrow_1}(A^\infty)$.

3.2.1 Properties of Algorithm 1

We explain the overview of the correctness argument for Algorithm 1 and some of its consequences.

Let us first prove the termination of the algorithm, which also yields a bound on the number of iterations of the main loop (thus on l_0 and i_0). Recall that for a given A, N_A denotes the number of pairs (q,r) where $q \in Q$ and r is an M-region (see Section 3.1).

▶ **Lemma 16.** *Let $\mathcal{D}(A)$ be a DTN region automaton returned by Algorithm 1. Then the slots of all region states in $\mathcal{D}(A)$ are bounded. Consequently, the number of iterations of Algorithm 1 is bounded by 2^{N_A+1}.*

The region automaton is of exponential size. Each iteration of Algorithm 1 takes exponential time since one might have to go through all region states in the worst case. By Lemma 16, the number of iterations is bounded by 2^{N_A}, which is doubly exponential in $|\mathcal{C}|$. Theorem 21 will show how to decide the reachability PMCP in exponential space.

We now prove the correctness of the algorithm in the following sense.

▶ **Theorem 17.** *Let A be a GTA, $\mathcal{D}(A) = (W, (\hat{q}, \hat{r}), \Sigma, E)$ its DTN region automaton, $\mathcal{S}(A)$ be the summary timed automaton. Then we have $\mathcal{L}(A^\infty)\downarrow_1 = \mathcal{L}(\mathcal{S}(A))$.*

To prove this, we need the following lemma that states a nontrivial property on which we rely: if a process can reach a given location q at global time t', then it can also reach q at any global time within the slot of t'. It follows that the set of global times at which a location can be occupied by at least one process is a union of intervals. Intuitively, this is why partitioning the region states by slots is a good enough abstraction in our setting.

▶ **Lemma 18.** *Consider a GTA A with bound function $M^\nearrow$. Let $\rho_r = (q_0, r_0) \xrightarrow{\sigma_0} \dots \xrightarrow{\sigma_{l-1}} (q_l, r_l)$ such that $(q_0, r_0) = (\hat{q}, \hat{r})$ be a computation in $\mathcal{R}_{M^\nearrow}(\mathsf{UG}(A))$ such that $\mathsf{slot}(r_l)$ is bounded. For all $t' \in \mathsf{slot}(r_l)$, there exists a timed computation $(q_0, v_0) \to \dots \to (q_l, v_l)$ in $\mathsf{UG}(A)$ such that $v_i \in r_i$ for $0 \le i \le l$, and $v_l(t) = t'$.*

The following lemma proves one direction of Theorem 17.

▶ **Lemma 19.** *Consider a trace $tt = (\delta_0, \sigma_0) \dots (\delta_{l-1}, \sigma_{l-1}) \in \mathcal{L}(\mathcal{S}(A))$. Let I be the unique interval of the form $[k, k]$ or $(k, k+1)$ with $k \in \mathbb{N}$ that contains $\delta_0 + \dots + \delta_{l-1}$.*

1. *For all $t' \in I$, there exists $n \in \mathbb{N}$, and a computation π of A^n such that $\mathsf{trace}(\pi)\downarrow_1 = (\delta'_0, \sigma_0) \dots (\delta'_{l-1}, \sigma_{l-1})$ for some $\delta'_i \ge 0$, and $\delta(\pi\downarrow_1) = t'$.*
2. *$tt \in \mathcal{L}(A^\infty)\downarrow_1$.*

The following lemma establishes the inclusion in the other direction. Given a computation π in A^n, we build a timed computation in $\mathcal{S}(A)$ on the same trace. Because the total time delay of π can be larger than the bound 2^{N_A}, we need to carefully calculate the slot in which they will end when projected to $\mathcal{S}(A)$.

▶ **Lemma 20.** *For any computation π of A^n with $n \in \mathbb{N}$, $\mathsf{trace}(\pi)\downarrow_1 \in \mathcal{L}(\mathcal{S}(A))$.*

Deciding the Reachability PMCP. It follows from Algorithm 1 that the reachability case can be decided in exponential space. This basically consists of running the main loop of Algorithm 1 without storing the whole list of all W_i, but only the last one. The loop needs to be repeated up to 2^{N_A+1} times (or until the target label σ_0 is found).

▶ **Theorem 21.** *The reachability PMCP for DTNs is decidable in EXPSPACE.*

Local Properties Involving Several Processes. The algorithm described above can be extended to compute $\mathcal{L}(A^\infty)\downarrow_{[1,a]}$. We define the *product* of k timed automata A_i, written $\otimes_{1 \le i \le k} A_i$, as the standard product of timed automata (see e.g., [16]) applied to A_i after replacing each label σ appearing in A_i by (i, σ).

▶ **Lemma 22.** *Given GTA A, and interval $I = [1, a]$, let $\mathcal{S}(A)$ be the summary automaton computed as above. Then $\mathcal{L}(A^\infty)\downarrow_I = \mathcal{L}(\otimes_{1 \le i \le a} \mathcal{S}(A))$.*

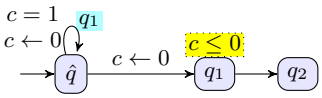


Figure 5 An example of GTA for which liveness is not preserved by our abstraction.

Limitations. Liveness properties (e.g., checking whether a transition can be taken an infinite number of times) are not preserved by our abstraction; since an infinite loop in the DTN region automaton may not correspond to a concrete computation in any A^n. In fact, consider the GTA in Figure 5. While there is an infinite loop on $\hat{q}$ in the DTN region automaton, no concrete execution takes the loop on $\hat{q}$ indefinitely, as each firing of this loop needs one more process to visit q_1, and then to leave it forever, due to the invariant $c \leq 0$.

4 Timed Lossy Broadcast Networks

Systems with lossy broadcast (a.k.a. "reconfigurable broadcast networks", where the underlying network topology might change arbitrarily at runtime) have received attention in the parameterized verification literature [22]. In the setting with finite-state processes, lossy broadcast is known to be at least as powerful as disjunctive guards, but it is unknown if it is strictly more powerful [15, Section 6]. We show that in our *timed* setting the two models are equally powerful, i.e., they simulate each other.

Lossy Broadcast Networks. Let Σ be a set of labels. A lossy broadcast timed automaton (LBTA) B is a tuple $(Q, \hat{q}, \mathcal{C}, \Sigma, \Lambda, \mathcal{T}, Inv)$ where $Q, \hat{q}, \mathcal{C}, Inv$ are as for TAs, and a transition is of the form $(q, g, \mathcal{C}_r, \sigma, \lambda, q')$, where $\lambda \in \{a!!, a?? \mid a \in \Lambda\}$. The *synchronization label* λ is used for defining global transitions. A transition with $\lambda = a!!$ is called a *sending transition*, and a transition with $\lambda = a??$ is called a *receiving transition*.

We also make Assumption 1 and Assumption 2 for LTBAs. The former means that the LBTA is timelock-free when all receiving transitions are removed. The semantics of a network of LBTAs is a timed transition system defined similarly as for NGTAs, except for *discrete transitions* which now induce a sequence of local transitions separated by 0 delays, as follows. Given $n > 0$, configurations of B^n are defined as for DTNs. Let $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ be a configuration of B^n. Consider indices $1 \leq i \leq n$ and $J \subseteq \{1, \ldots, n\} \setminus \{i\}$, and labels $\sigma, \sigma_j \in \Sigma$ for $j \in J$, such that

i) $(q_i, v_i) \xrightarrow{a!!,\sigma} (q_i', v_i')$ is a sending transition of B, and

ii) for all $j \in J$: $(q_j, v_j) \xrightarrow{a??,\sigma_j} (q_j', v_j')$ is a receiving transition of B.

Then the timed transition system of B^n contains the transition sequence $\mathfrak{c} \xrightarrow{(0,(i,\sigma))} \mathfrak{c}' \xrightarrow{(0,(j_1,\sigma_{j_1}))} \mathfrak{c}_1' \xrightarrow{(0,(j_2,\sigma_{j_2}))} \cdots \xrightarrow{(0,(j_m,\sigma_{j_m}))} \mathfrak{c}_m'$ for all possible sequences $j_1, \ldots, j_m$ where $J = \{j_1, \ldots, j_m\}$. Non-zero delays only occur outside of these chains of 0-delay transitions. For a given LBTA B, the family of systems B^∞ is called a *lossy broadcast timed network* (LBTN).

Simulating LBTN by DTN (and vice versa). The following theorem states that LBTAs and GTAs are inter-reducible.

▶ **Theorem 23.** *For all GTA A, there exists an LBTA B s.t. $\forall k{\geq}1$, $\mathcal{L}(A^\infty){\downarrow_{[1,k]}} \equiv \mathcal{L}(B^\infty){\downarrow_{[1,k]}}$. For all LBTA B, there exists a GTA A s.t. $\forall k{\geq}1, \mathcal{L}(A^\infty){\downarrow_{[1,k]}} = \mathcal{L}(B^\infty){\downarrow_{[1,k]}}$.*

Sketch. Simulation of disjunctive guards by lossy broadcast is simple: a transition from q to q' with location guard γ is simulated in lossy broadcast by the sender taking a self-loop transition on γ, and the receiver having a synchronizing transition from q to q'.

The other direction is where we need the power of clock invariants: to simulate a lossy broadcast where the sender moves from q to q' and a receiver moves from q_{rcv} to q'_{rcv}, in the DTN we first let the sender move to an auxiliary location q_σ (from which it can later move on to q'), and have a transition from q_{rcv} to q'_{rcv} that is guarded with q_σ. To ensure that no time passes between the steps of sender and receiver, we add an auxiliary clock c_{snd} that is reset when moving into q_σ, and q_σ has clock invariant $c_{\mathsf{snd}} = 0$.

In both directions, auxiliary transitions that are only needed for the simulation are labeled with fresh symbols in Σ^- such that they do not appear in the language of the system. ◀

Because the reduction to DTNs is in linear-time, we get the following.

▶ **Corollary 24.** *The reachability PMCP for LBTN is decidable in EXPSPACE.*

5 Synchronizing Timed Networks and Timed Petri Nets

We first introduce synchronizing timed networks. Our definitions follow [6, 5], except that their model considers systems with a controller process, whereas we assume (like in our previous models) that all processes execute the same automaton.

Synchronizing Timed Network. A *synchronizing timed automaton* (STA) $\mathcal{S}$ is a tuple $(Q, \hat{q}, \mathcal{C}, Inv, \mathcal{R})$ where Q, $\hat{q}$, $\mathcal{C}$, Inv are as for TAs, and $\mathcal{R}$ is a finite set of *rules*, where each rule $r \in \mathcal{R}$ is of the form $\langle q_{r,1} \xrightarrow{g_{r,1},\mathcal{C}_{r,1},\sigma_{r,1}} q'_{r,1}, \cdots, q_{r,m} \xrightarrow{g_{r,m},\mathcal{C}_{r,m},\sigma_{r,m}} q'_{r,m} \rangle$ for some $m \in \mathbb{N}$ and with $(q_i, g_{r,i}, \mathcal{C}_{r,i}, \sigma_{r,i}, q'_i) \in Q \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times \Sigma \times Q$ for $1 \leq i \leq m$.

The semantics of a *network of STAs* (NSTA) is defined as for NGTAs, except for *discrete transitions*, which now synchronize a subset of all processes in the following way: Let $r \in \mathcal{R}$ be a rule (of the form described above) and $\mathfrak{c} = \big((q_1, v_1), \ldots, (q_n, v_n)\big)$ a configuration of $\mathcal{S}^n$. Assume

i) there exists an injection $h : \{1, \ldots, m\} \to \{1 \ldots n\}$ such that for each $1 \leq i \leq m$, $q_{h(i)} = q_{r,i}$, $q_{h(i)} \xrightarrow{g_{r,i},\mathcal{C}_{r,i},\sigma_{r,i}} q'_{h(i)}$ is an element of r, $v_{h(i)} \models g_{r,i}$ and $v'_{h(i)} = v_{h(i)}[\mathcal{C}_{r,i} \leftarrow 0]$, and

ii) $j \notin \mathrm{range}(h)$, $q'_j = q_j$ and $v'_j = v_j$.

Then the timed transition system of $\mathcal{S}^n$ contains the transition sequence $\mathfrak{c} \xrightarrow{(0,(h(1),\sigma_{r,1}))} \mathfrak{c}_1 \xrightarrow{(0,(h(2),\sigma_{r,2}))} \ldots \xrightarrow{(0,(h(m),\sigma_{r,m}))} \mathfrak{c}_m$. That is, m distinct processes take individual transitions according to the rule without delay, and the configurations of the non-participating processes remain unchanged.

Again, we also make Assumptions 1 and 2 for STAs. The former means here that $\mathcal{S}$ is timelock-free when all transitions of rules with $m > 1$ are removed. All other notions follow in the natural way. Given an STA $\mathcal{S}$, the family of systems $\mathcal{S}^\infty$ is called a *synchronizing timed network* (STN).

▶ **Theorem 25.** *For all GTA A with set of locations Q, there exists an STA $\mathcal{S}$ with set of locations Q such that for every $q \in Q$: q is reachable in A iff q is reachable in $\mathcal{S}$. For all STA $\mathcal{S}$ with set of locations $Q_\mathcal{S}$, there exists a GTA A with set of locations $Q_A \supseteq Q_\mathcal{S}$ such that for every $q \in Q_\mathcal{S}$: q is reachable in A iff q is reachable in $\mathcal{S}$.*

Sketch. Simulation of disjunctive guards by STAs is simple: a transition from q to q' with location guard γ is simulated by a pairwise synchronization, where one process takes a self-loop on γ, and the other moves from q to q'.

Table 1 Existing and new decidability results for location reachability (Reach) and local trace properties (Trace) for DTN, LBTN, and STN with a single ($|\mathcal{C}| = 1$) or multiple clocks ($|\mathcal{C}| > 1$), and with (*Inv*) or without invariants (~~*Inv*~~). Entries with ✓* need to satisfy Assumption 1.

	DTN			LBTN			STN		
	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|=1$	$\|\mathcal{C}\|>1$	$\|\mathcal{C}\|>1$
	~~*Inv*~~	~~*Inv*~~	*Inv*	~~*Inv*~~	~~*Inv*~~	*Inv*	~~*Inv*~~	~~*Inv*~~	*Inv*
Reach	✓[31]	✓[31]	✓	✓[6, 11]	✓	✓	✓[6]	✓	✓
Trace	✓[31]	✓[31]	✓*	✓	✓	✓*	?	?	?

Conversely, to simulate a rule r of the STA with m participating processes, we add auxiliary locations $p_{r,i}$, for $1 \leq i \leq m$, each with a clock invariant (on an additional clock only used for the simulation) that ensures that no time passes during simulation. For each element $q_{r,i} \xrightarrow{g_{r,i},\mathcal{C}_{r,i},\sigma_{r,i}} q'_{r,i}$ of r, we have a transition from $q_{r,i}$ to $p_{r,i}$, and from there to $q'_{r,i}$. A transition to $p_{r,i}$ is guarded with $p_{r,i-1}$ (except when $i = 1$), and with the clock constraint $g_{r,i}$, and all transitions to $q'_{r,i}$ are guarded with $p_{r,m}$. This ensures that any $q'_{r,i}$ is reachable through this construction if and only if the global configuration at the beginning would allow the STA to execute rule r. To avoid introducing timelocks, each of the $p_{r,i}$ has an additional transition with a trivial location guard and no clock guard to a new sink location $q_\perp$ that does not have an invariant. I.e., if simulation of a rule is started but cannot be completed (because there are processes in some but not all of the locations $q_{r,i}$), then processes can (and have to) move to $q_\perp$. ◀

▶ **Corollary 26.** *The reachability PMCP for STN is decidable in EXPSPACE.*

Note that the construction in our proof is in general not suitable for language equivalence, i.e., $\mathcal{L}(A^\infty)\downarrow_{[1,k]}$ might contain traces that are not in $\mathcal{L}(\mathcal{S}^\infty)\downarrow_{[1,k]}$.

Abdulla et al. [2] considered the *universal safety problem* of *timed Petri nets* – that is, whether a given transition can eventually be fired for any number of tokens in the initial place – and solved it for the case where each token has a single clock. The question whether the problem is decidable for tokens with multiple clocks remained open. This problem, in the multi-clock setting, can be reduced to the PCMP of STNs. The reduction is conceptually straightforward and computable in polynomial time in the size of the input.

▶ **Corollary 27.** *The universal safety problem for timed Petri nets with an arbitrary number of clocks is decidable in EXPSPACE.*

6 Conclusion

In this paper, we solved positively the parameterized model checking problem (PMCP) for finite local trace properties of disjunctive timed networks (DTNs) with invariants. We also proved that the PMCP for networks that communicate via lossy broadcast can be reduced to the PMCP for DTNs, and is therefore decidable. Additional results also allowed us to solve positively the open problem from [2] whether the universal safety problem for timed Petri nets with multiple clocks is decidable. Table 1 gives an overview of our results, compared to existing results for the classes of systems we consider.

In addition to the results presented here, we believe that our proof techniques can be extended to support timed networks with more powerful communication primitives, and in some cases to networks with controllers.

Future work will include tightening the complexity bounds for the problems considered here, as well as the development of zone-based algorithms that can be more efficient in practice than a direct implementation of the algorithms presented here.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Timed lossy channel systems. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 374–386. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.FSTTCS.2012.374.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Radu Ciobanu, Richard Mayr, and Patrick Totzke. Universal safety for timed Petri nets is PSPACE-complete. In Sven Schewe and Lijun Zhang, editors, *CONCUR*, volume 118 of *LIPIcs*, pages 6:1–6:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CONCUR.2018.6.

3 Parosh Aziz Abdulla and Giorgio Delzanno. Parameterized verification. *International Journal on Software Tools for Technology Transfer*, 18(5):469–473, 2016. doi:10.1007/s10009-016-0424-3.

4 Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612:1–22, 2016. doi:10.1016/j.tcs.2015.07.048.

5 Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *LiCS*, pages 345–354. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319629.

6 Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003. doi:10.1016/S0304-3975(01)00330-9.

7 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In Edmund M Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 685–725. Springer, 2018. doi:10.1007/978-3-319-10575-8_21.

8 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.

9 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Computing*, 31(3):187–222, 2018. doi:10.1007/s00446-017-0302-6.

10 Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2015. doi:10.1007/978-3-662-47666-6_30.

11 Étienne André, Benoît Delahaye, Paulin Fournier, and Didier Lime. Parametric timed broadcast protocols. In Constantin Enea and Ruzica Piskac, editors, *VMCAI*, volume 11388 of *Lecture Notes in Computer Science*, pages 491–512. Springer, 2019. doi:10.1007/978-3-030-11245-5_23.

12 Étienne André, Paul Eichler, Swen Jacobs, and Shyam Lal Karra. Parameterized verification of disjunctive timed networks. In Rayna Dimitrova and Ori Lahav, editors, *VMCAI*, volume 14499 of *Lecture Notes in Computer Science*, pages 124–146. Springer, 2024. doi:10.1007/978-3-031-50524-9_6.

13 E. André, S. Jacobs, S.L. Karra, and O. Sankur. Parameterized verification of timed networks with clock invariants (extended version), 2024. arXiv:2408.05190. URL: https://arxiv.org/abs/2408.05190.

14 Krzysztof R. Apt and Dexter Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986. doi:10.1016/0020-0190(86)90071-2.

15 A. R. Balasubramanian and Chana Weil-Kennedy. Reconfigurable broadcast networks and asynchronous shared-memory systems are equivalent. In Pierre Ganty and Davide Bresolin, editors, *GandALF*, volume 346 of *EPTCS*, pages 18–34, 2021. doi:10.4204/EPTCS.346.2.

16 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3.

17 Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In Anil Seth and Nisheeth K. Vishnoi, editors, *FSTTCS*, volume 24 of *LIPIcs*, pages 501–513. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.501.

18 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.

19 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000. doi:10.1007/10722167_35.

20 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, August 2004. doi:10.1016/j.tcs.2004.04.003.

21 Patricia Bouyer, Uli Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, Joël Ouaknine, and James Worrell. Model checking real-time systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1001–1046. Springer, 2018. doi:10.1007/978-3-319-10575-8_29.

22 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2010. doi:10.1007/978-3-642-15375-4_22.

23 E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In David A. McAllester, editor, *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000. doi:10.1007/10721959_19.

24 E. Allen Emerson and Kedar S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4):527–550, 2003. doi:10.1142/S0129054103001881.

25 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LiCS*, pages 352–359. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782630.

26 Javier Esparza, Stefan Jaax, Mikhail A. Raskin, and Chana Weil-Kennedy. The complexity of verifying population protocols. *Distributed Computing*, 34(2):133–177, 2021. doi:10.1007/s00446-021-00390-x.

27 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

28 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

29 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. doi:10.1006/inco.1994.1045.

30 Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In Supratik Chakraborty and Amit Kumar, editors, *FSTTCS*, volume 13 of *LIPIcs*, pages 78–89. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.78.

31 Luca Spalazzi and Francesco Spegni. Parameterized model checking of networks of timed automata with Boolean guards. *Theoretical Computer Science*, 813:248–269, 2020. doi:10.1016/j.tcs.2019.12.026.

32 Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, 1988. doi:10.1016/0020-0190(88)90211-6.

ε-Stationary Nash Equilibria in Multi-Player Stochastic Graph Games

Ali Asadi
Institute of Science and Technology Austria, Klosterneuburg, Austria

Léonard Brice
Université Libre de Bruxelles, Belgium

Krishnendu Chatterjee
Institute of Science and Technology Austria, Klosterneuburg, Austria

K. S. Thejaswini
Institute of Science and Technology Austria, Klosterneuburg, Austria

Abstract

A strategy profile in a multi-player game is a Nash equilibrium if no player can unilaterally deviate to achieve a strictly better payoff. A profile is an ε-Nash equilibrium if no player can gain more than ε by unilaterally deviating from their strategy. In this work, we use ε-Nash equilibria to approximate the computation of Nash equilibria. Specifically, we focus on turn-based, multiplayer stochastic games played on graphs, where players are restricted to stationary strategies – strategies that use randomness but not memory.

The problem of deciding the constrained existence of stationary Nash equilibria – where each player's payoff must lie within a given interval – is known to be $\exists\mathbb{R}$-complete in such a setting (Hansen and Sølvsten, 2020). We extend this line of work to stationary ε-Nash equilibria and present an algorithm that solves the following promise problem: given a game with a Nash equilibrium satisfying the constraints, compute an ε-Nash equilibrium that *ε-satisfies* those same constraints – satisfies the constraints up to an ε additive error. Our algorithm runs in $\mathsf{FNP}^{\mathsf{NP}}$ time.

To achieve this, we first show that if a constrained Nash equilibrium exists, then one exists where the non-zero probabilities are at least an inverse of a double-exponential in the input. We further prove that such a strategy can be encoded using floating-point representations, as in the work of Frederiksen and Miltersen (2013), which finally gives us our $\mathsf{FNP}^{\mathsf{NP}}$ algorithm.

We further show that the decision version of the promise problem is NP-hard. Finally, we show a partial tightness result by proving a lower bound for such techniques: if a constrained Nash equilibrium exists, then there must be one where the probabilities in the strategies are double-exponentially small.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Nash Equilibria, ε-Nash equilibria, Approximation, Existential Theory of Reals

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.9

Related Version *Full Version*: `arXiv:2508.15356` [2]

Funding This work is a part of project VAMOS that has received funding from the European Research Council (ERC), grant agreement No 101020093.

1 Introduction

Modelling decentralised systems involving multiple agents requires understanding the interactions between different players, each with their own objectives. Stochastic games – graphs in which some nodes are controlled by agents and others are controlled by a random


© Ali Asadi, Léonard Brice, Krishnendu Chatterjee, and K. S. Thejaswini;
licensed under Creative Commons License CC-BY 4.0



environment – provide a modelling framework for a wide range of domains, including epidemic processes [17], formal verification [13], learning theory [1], cyber-physical systems [20], distributed and probabilistic programs [10], and probabilistic planning [22].

These stochastic multi-player turn-based games are a specific class of models that capture interactions between multiple agents on such graph arenas. These games have been extensively studied, with several results characterizing their computational complexity [8, 24] and others studying restrictions and different payoff functions that yield tractable fragments [24, 7]. While it is known that Nash equilibria (NEs) always exist in these settings, the equilibria that arise in proofs are often adversarial in nature, offering poor outcomes for all players [8, 23]. A more compelling question is whether one can decide, and compute if possible, a Nash Equilibrium that satisfies certain payoff constraints. However, this problem becomes undecidable when players are allowed unbounded memory. Indeed, it is already known that even checking whether a player can achieve a payoff within a given interval is undecidable with just 10 agents [24, Theorem 4.9]. Restricting players to deterministic (pure) strategies does not help, as the problem remains undecidable. Thus, for any hope of tractability, we must restrict each player to use bounded memory. In practice, this is also desirable: strategies requiring large memory lead to complex controllers and are harder to implement. Fortunately, finite-memory strategies can be encoded directly in the state space of the game, and in many cases, it suffices to consider stationary strategies, which require no memory at all.

In this work, we only focus on Nash equilibria where each player is restricted to stationary strategies. Indeed, henceforth, whenever we mention strategies, we only mean stationary strategies, unless specified otherwise. However, it is known that the constrained existence problem even when restricted to such stationary strategies is $\exists\mathbb{R}$-complete [16]. Moreover, for the functional version of the problem, the probabilities used to indicate the distribution of these stationary strategies can have irrational values [24, Theorem 4.6] even for games with just four players. Indeed, such strategies cannot be executed in real-world scenarios since it is hard to simulate choosing an edge with such irrational probabilities.

The standard definition of Nash equilibrium requires that no player can improve their payoff by any non-zero amount by unilaterally deviating from their strategy even if the improvement is arbitrarily small [19]. This strict notion of optimality contributes significantly to the computational complexity of deciding whether a constrained Nash equilibrium exists. From a practical perspective, however, it is reasonable to assume that players are unlikely to deviate from their current strategies for only negligible gains.

Our setting. Motivated by this discussion, we consider the relaxed variant of the equilibrium concept that is also well-studied [18, 9]. Specifically, we study the problem of computing ε-approximate Nash equilibria – in which the expected utility of each player is required to be within ε of the optimum response to the other players' strategies – while still satisfying given constraints on the players' payoffs. Formally, we focus on multi-player turn-based games with terminal rewards and for a (stationary) strategy profile that is an ε-NE that also approximately satisfies the constraints (the constraints are also satisfied up to an ε-additive factor), assuming there is an exact stationary NE.

The decision version of this problem is defined as follows: given such a game, determine whether either (i) there exists an exact NE that satisfies the constraints, or (ii) no ε-approximate NE satisfies the constraints ε-approximately. Instances that fall outside these two cases – that is, where an ε-approximate equilibrium exists but no exact one does – are considered indeterminate, and the output may be arbitrary in those cases. Note that whenever there is an exact NE, by definition, there exists an ε-NE.

Our results. We show that this approximate variant is computationally more tractable than the general problem of deciding the existence of a stationary constrained equilibrium. In particular, we present an algorithm that solves the problem within the class FNP with access to an NP oracle. We also show that the probabilities required for the exact version of the problem can be double-exponentially small, proving some evidence of hardness of the problem. Finally, we show that the decision version of the problem is NP-hard.

Technical overview

Upper bound. Recall our result that the problem lies in $\mathsf{FNP}^{\mathsf{NP}}$. Our approach relies on showing that if a stationary equilibrium exists, then there is one in which the probability values used in the strategies are not too small – specifically, they are lower-bounded by the inverse of a doubly exponential function of the input size. This structural property is established using techniques from the work of Hansen, Koucký, and Miltersen on zero-sum concurrent reachability games [15]. However, this bound alone is insufficient, because representing such tiny probabilities explicitly requires at least exponentially many bits to represent using fixed-point presentation.

To represent the probabilities used in strategies more succinctly than with fixed-point representation, we use floating-point values, a standard tool in numerical analysis. While floating-point representations cannot express the exact probabilities required in an exact Nash equilibrium, they are sufficient for representing approximate equilibria. We prove (see Lemma 9) that if an exact Nash equilibrium exists, then there is also an ε-NE that can be encoded using floating-point numbers.

The proof relies on two ideas from the work of Frederickson and Milterson [14]. First, any probability distribution can be approximated by one using floating-point values, such that the difference between them is small in a precise sense. Second, when all the probabilities in a Markov chain are approximated in this way, the expected value of the chain changes by at most a polynomial function of the approximation error. To formalise this notion of approximation, a distance metric introduced by Solan [21] is used to show how values of Markov chains behave under small perturbations.

Using these results, we design an algorithm for the problem. The algorithm first guesses a stationary approximate equilibrium that satisfies the payoff constraints up to the given error. Our earlier results ensure that such a guess can be represented using floating-point numbers with only polynomially many bits. The algorithm then verifies that this strategy profile is indeed an approximate equilibrium by checking, for each player, whether any alternative strategy would improve their payoff by more than the allowed margin of error. This check reduces to solving a Markov decision process for each player, where the strategies of the other players are fixed. We show that solving this single-player game can be done with a single call to an NP oracle for each player (see Lemma 13). Therefore, the search problem lies in the class $\mathsf{FNP}^{\mathsf{NP}}$, and the corresponding decision problem is in $\mathsf{NP}^{\mathsf{NP}}$.

Small probabilities. We first consider the limits of the above approach. Prior work by Deligkas, Fearnley, Melissourgos, and Spirakis [11] observed that the class ε-$\exists\mathbb{R}$, which captures the complexity of deciding whether approximate solutions exist to systems of real-valued constraints, is polynomial time reducible to the class $\exists\mathbb{R}$ itself. This equivalence in the complexity of $\exists\mathbb{R}$ and its approximation version hinges on the fact that solutions to such existential theory of the reals (ETR) problems can involve values that are double-

exponentially large. To address this, they identified a restricted fragment of ETR for which ε-approximate solutions can be computed using a quasi-polynomial time approximation scheme (QPTAS).

In our setting, we use a key structural result (see Lemma 16) showing that stationary Nash equilibria, when they exist, can involve probabilities that are double-exponentially small, but not smaller. One might suspect that such small probabilities are merely theoretical and not actually be needed to represent probabilities occurring in such equilibria. However, in Section 4, we present a concrete example with just five players where double-exponentially small probabilities are indeed necessary for an exact equilibrium to exist. We note that previous results establishing $\exists\mathbb{R}$-completeness for related problems required seven players, underscoring the tightness of our example.

Hardness results. Finally, in Section 5, we also show that even the decision version of the promise problem is NP-hard, via a reduction from the classical 3SAT problem. While it was already known that the constrained existence of stationary Nash equilibria is NP-hard [24, Theorem 4.4] and later shown to be even $\exists\mathbb{R}$-complete, we prove that even the relaxed version of the problem – deciding whether an approximate (i.e., ε-close) stationary Nash equilibrium exists that satisfies the given constraints – is also NP-hard.

Related work. The approximation techniques we use to establish our $\mathsf{FNP}^{\mathsf{NP}}$ upper bound have appeared in various forms in the literature, particularly in the context of approximating values in concurrent games. For example, Frederiksen and Miltersen [14] introduced similar floating-point approximation methods to compute values of concurrent reachability games up to arbitrary precision. Their work placed the problem in the complexity class $\mathsf{TFNP}^{\mathsf{NP}}$. However, for this class of games, no corresponding hardness results – such as the NP-hardness we establish for our setting – were previously known.

These techniques have also been extended to other objectives in the same setting of concurrent games. In particular, Asadi, Chatterjee, Saona, and Svoboda [3] applied related methods to show $\mathsf{TFNP}^{\mathsf{NP}}$ upper bounds for games with stateful-discounted objectives and parity objectives.

More recently, Bose, Ibsen-Jensen, and Totzke [6] studied ε-approximate Nash equilibria in concurrent and partial-observation settings, also using approximation-based techniques. However, their work differs from ours in two key ways. First, they do not address the *constrained existence problem* for ε-Nash equilibria. We note that the unconstrained variant of the problem, which they consider, is significantly easier in terms of computational complexity. Second, they assume the number of players is fixed, whereas in our setting, the number of players is part of the input. To our knowledge, our work is the first to analyse constrained ε-Nash equilibria in multi-player games where the number of players is not fixed.

2 Preliminaries

We assume that the reader is familiar with the basics of probability and graph theory. However, we define some concepts for establishing notation.

Probabilities. Given a (finite or infinite) set of outcomes Ω and a probability measure $\mathbb{P}$ over Ω, let X be a random variable over Ω, i.e., a mapping $X : \Omega \to \mathbb{R}$. We write $\mathbb{E}^{\mathbb{P}}[X]$, or simply $\mathbb{E}[X]$, for the expected value of X, when defined. Given a finite set S, a *probability distribution* over S is a mapping $d : S \to [0, 1]$ that satisfies the equality $\sum_{x \in S} d(x) = 1$.

Size of numbers. Integers are always implicitly assumed to be represented with their binary encoding. For a given integer $n \in \mathbb{N}$, we therefore write $\mathsf{bit}(n) = \lceil \log_2(n+1) \rceil$ for the number of bits required to write n. Similarly, unless stated otherwise, a rational number $\frac{p}{q}$, where p and q are co-prime, is represented by the pair of the encodings of p and q, and we write $\mathsf{bit}\left(\frac{p}{q}\right) = \mathsf{bit}(p) + \mathsf{bit}(q) + 1$.

Graphs and games. A directed graph (V, E) consists of a set V of *vertices* and a set E of ordered pairs of vertices, called *edges*. For simplicity, we often write uv for an edge $(u, v) \in E$. A *path* in the directed graph (V, E) is a (finite or infinite) ordered sequence of vertices from V such that every pair of two consecutive elements is an edge. A *cycle* is a path with distinct vertices such that the last element of the sequence and the first one also forms an edge.

Throughout this paper, we use the word *game* for multiplayer simple quantitative turn-based games played on graphs.

▶ **Definition 1** (Game). *A* game *is a tuple $\mathcal{G}$ that consists of:*

- *a directed graph (V, E), called the* underlying *graph of $\mathcal{G}$;*
- *a finite set Π of* players*;*
- *a partition $(V_i)_{i \in \Pi \cup \{?\}}$ of the set V, where V_i denotes the set of vertices* controlled *by player i, and the vertices in $V_?$ are called* stochastic vertices*;*
- *a* probability function $\mathsf{p} : E(V_?) \to [0, 1]$*, such that for each stochastic vertex s, the restriction of p to $E(s)$ is a probability distribution;*
- *a mapping $\mu : T \to [0, 1]^\Pi$ called* payoff function*, where T is the set of* terminal vertices*, that is, vertices of the graph (V, E) that have no outgoing edges. We also write μ_i, for each player i, for the function that maps a terminal vertex t to the i^{th} coordinate of the tuple $\mu(t)$.*
- *a vertex $v_0 \in V$, which is the* initial *vertex of the game.*

When referring to a game $\mathcal{G}$, we will often use the notations V for vertices, E for edges, Π for players, and so on without necessarily recalling them.

▶ **Definition 2** (Markov decision process, Markov chain). *A* Markov decision process *is a game with one player. A* Markov chain *is a game with zero players.*

Plays. We call *play* a path in the underlying graph that is infinite, or whose last vertex is a terminal vertex. The payoff functions μ and μ_i, defined only for terminal vertices, are extended naturally to plays as follows: for a play of the form ht, with $t \in T$, we define $\mu(ht) = \mu(t)$, and for an infinite play π, we define $\mu(\pi) = (0)_{i \in \Pi}$ (all players receive the payoff 0 in an infinite path where no terminal vertex is reached).

Strategies, and strategy profiles. In this paper, by *strategy*, we mean a stationary strategy, that is, a strategy that only depends on the current vertex.

Thus, in a game $\mathcal{G}$, a *strategy* for player i is a mapping σ_i that maps each vertex $v \in V_i$ to a probability distribution over the neighbours of v. A *strategy profile* for a subset $P \subseteq \Pi$ is a tuple $(\sigma_i)_{i \in P}$, that we usually write $\bar{\sigma}$ if $P = \Pi$, and $\bar{\sigma}_{-i}$ if $P = \Pi \backslash \{i\}$ for some player i. We then write (σ_{-i}, σ'_i) to denote the strategy profile $\bar{\tau}$ defined by $\tau_i = \sigma'_i$ and $\tau_j = \sigma_j$ for $j \neq i$.

A strategy profile $\bar{\sigma}$ for all players in the game $\mathcal{G}$ defines a probability measure $\mathbb{P}_{\bar{\sigma}}$ over plays – which turns the payoff functions μ_i into random variables. We then write $\mathbb{E}(\bar{\sigma})$ for the expectation operator $\mathbb{E}^{\mathbb{P}_{\bar{\sigma}}}$. If the game is a Markov chain, then we just write $\mathbb{E}$ to represent

this value. We say the maximum payoff of an MDP is the maximum of all expectations over all stationary strategies and write $\mathsf{val} = \max_\sigma \mathbb{E}(\bar{\sigma})$ to denote it.

Equilibria. In such games, we study *equilibria*, that is, strategy profiles that offer some stability guarantees. The most famous notion of equilibrium is the *Nash equilibrium* [19].

▶ **Definition 3** (Nash equilibrium). *A* Nash equilibrium*, or* NE *for short, in the game $\mathcal{G}$ is a strategy profile $\bar{\sigma}$ such that for each player i and every strategy σ'_i, we have*

$$\mathbb{E}(\bar{\sigma}_{-i}, \sigma'_i)[\mu_i] \leqslant \mathbb{E}(\bar{\sigma})[\mu_i].$$

A classical quantitative relaxation of Nash equilibria is that of ε-approximate Nash equilibria, or just ε-Nash equilibria.

▶ **Definition 4** (ε-Nash equilibrium). *Let $\varepsilon > 0$. An* ε-Nash equilibrium*, or* ε-NE *for short, in the game $\mathcal{G}$ is a strategy profile $\bar{\sigma}$ such that for each player i and every strategy σ'_i, we have $\mathbb{E}(\bar{\sigma}_{-i}, \sigma'_i)[\mu_i] - \varepsilon \leqslant \mathbb{E}(\bar{\sigma})[\mu_i]$.*

Problem. A classical decision problem about Nash equilibria is the following one.

▶ **Problem 1** (Constrained existence problem of Nash equilibria). *Given a game $\mathcal{G}$ and two vectors $\bar{x}, \bar{y} \in \mathbb{Q}^\Pi$, called* threshold vectors*, does there exist an NE $\bar{\sigma}$ in $\mathcal{G}$ that satisfies the inequality $\bar{x} \leqslant \mathbb{E}(\bar{\sigma})[\mu] \leqslant \bar{y}$?*

Problem 1 is known to be undecidable if the players are allowed to use memory [24], which is excluded by our formalism here. Where players are restricted to *stationary* strategy – as in our definition – this problem is known to be $\exists\mathbb{R}$-complete [16], where $\exists\mathbb{R}$ is the complexity class of problems that can be reduced to the satisfiability of a formula in the existential theory of the reals. In order to obtain a more tractable problem, we consider here an approximated version defined using the notion of ε-NEs: we wish to obtain, for sure, the answer *yes* when an NE satisfying the constraint exists, and to obtain, for sure, the answer *no* when we are far from having such an NE because there is not even an *ε-NE* that satisfies the constraint *up to ε* – but on the limit case, we can accept any answer. Formally, we want an algorithm that solves the following problem.

▶ **Problem 2** (Approximate constrained existence problem of Nash equilibria). *Given a game $\mathcal{G}$, two vectors $\bar{x}, \bar{y} \in \mathbb{Q}^\Pi$, and a rational number $\varepsilon > 0$ input using fixed point representation, such that either:*

1. *there is an NE $\bar{\sigma}$ in $\mathcal{G}$ with $x_i \leqslant \mathbb{E}(\bar{\sigma})[\mu_i] \leqslant y_i$ for each player i; or*
2. *there is no ε-NE in $\mathcal{G}$ satisfying $x_i - \varepsilon \leqslant \mathbb{E}(\bar{\sigma})[\mu_i] \leqslant y_i + \varepsilon$ for each player i;*

holds, then are we in case 1?

We call *functional* approximate constrained NE problem the functional version of this problem, in which the answer *yes* is replaced by a succinct representation of an ε-NE that satisfies the constraint up to ε. In the sequel, we show a $\mathsf{FNP}^{\mathsf{NP}}$ upper bound on this functional problem, which implies an $\mathsf{NP}^{\mathsf{NP}}$ upper bound on the non-functional one; and an NP lower bound on the non-functional approximated problem, which implies an FNP lower bound on the functional one.

3 Finding ε-Nash equilibria

In this section, we present the $\mathsf{FNP}^{\mathsf{NP}}$ upper bound for the functional approximate constrained NE problem. We always use n to represent the number of vertices in the game and m to represent the number of edges in the game, and τ which corresponds to the bit size used to represent the probabilities.

Throughout this section, we assume that we are given an instance $(\mathcal{G}, \bar{x}, \bar{y}, \varepsilon)$ of the functional approximate constrained problem. We call *(ε-)constrained NE* an (ε-)NE in $\mathcal{G}$ such that each player i's payoff lies in the interval $[x_i, y_i]$ (resp. $[x_i - \varepsilon, y_i + \varepsilon]$). The rest of this section is dedicated to the proof of the following theorem.

▶ **Theorem 5.** *There exists an* $\mathsf{FNP}^{\mathsf{NP}}$ *procedure to solve the functional approximated constrained problem of NEs.*

We begin by defining the existential theory of the reals and showing how the problem of computing stationary Nash equilibria can be encoded using sentences in it.

By applying a result of Basu, Pollack, and Roy [5] (restated in Lemma 8), it can be shown that any solution to the resulting system of polynomial inequalities lies within a ball of at most double-exponential radius. This implies that, in the worst case, the smallest non-zero probabilities in a stationary equilibrium may be as small as inverse double-exponential in the size of the input.

To represent such small probabilities succinctly, we introduce a floating-point representation that allows encoding of some double-exponentially small values using only polynomially many bits. Next, using a result of Solan [21] (see Lemma 11), we observe that small perturbations in transition probabilities lead to only small changes in the value of the associated Markov chain. This continuity result ensures that approximate equilibria obtained where probabilities are slightly perturbed still yield approximately correct values. This shows that it is enough to represent the probabilities using floating-point numbers with polynomial precision.

Finally, we provide a coNP procedure (see Lemma 13) to decide whether the maximum payoff in a Markov Decision Process (MDP), where probabilities are given in floating-point representation, lies above or below a given threshold up to an additive ε.

Together, these components give us our desired $\mathsf{FNP}^{\mathsf{NP}}$ procedure: we guess a stationary strategy profile that approximately satisfies the constraints and verify it using the above results, and use the fact that such strategies admit succinct polynomial-size representations.

Existential theory of the reals. A sentence in the *existential theory of the reals* is of the form

$$\varphi \;=\; \exists x_1\, \exists x_2 \;\cdots\; \exists x_k\; F(x_1, \ldots, x_k)\,,$$

where $F(x_1, \ldots, x_k)$ is a *quantifier-free* formula in the language of ordered fields over the reals $\mathbb{R}$. More concretely, atomic sub-formulas are

$$p(x_1, \ldots, x_k) = 0 \quad \text{or} \quad p(x_1, \ldots, x_k) < 0,$$

where $p \in \mathbb{R}[x_1, \ldots, x_k]$ is a polynomial over $\mathbb{R}$, and arbitrary formulas are built by Boolean connectives.

▶ **Lemma 6.** *If there exists a constrained Nash equilibrium $\bar{\sigma}$, then there exists a constrained Nash equilibrium such that the non-zero probabilities are at least* $2^{-\tau 2^{O(m|\Pi|)}}$.

Proof. This statement is similar to a result in the work of Hansen, Koucký, and Miltersen [15, Thm. 4]. We write the ETR sentence for solving constrained stationary Nash equilibrium in the following proposition.

▶ **Proposition 7.** *There exists an ETR sentence for computing constrained stationary Nash equilibrium with $O(m|\Pi|)$ variables, where all polynomials are of degree at most 2.*

Proof. The following part is similar to the ETR sentence for constrained stationary Nash equilibria found in the work of Ummels and Wojtczak [24, Theorem 4.5]. However, we require the explicit bounds on the number of variables, the degree polynomials, and the length of the sentence. Thus, we rewrite the explicit ETR sentence. First, by the assumption, there exists a strategy profile that is a stationary Nash equilibrium $\bar{\sigma}$. Let S be the set of all non-zero edges in the strategy profile $\bar{\sigma}$. Furthermore, we define the set of vertices V_S from which terminals can be reached with non-zero probability. Second, we have a formula that states that the variables p_{vw} indeed describe a strategy. We enforce that the non-zero variables belong to the set S. We further ensure that for stochastic vertices, the variable p_{vw} encodes exactly the value dictated by the probability function p by the stochastic vertex. This involves satisfying the following constraints:

1. $p_{vw} > 0$; for each $vw \in S$
2. $p_{vw} \leqslant 1$; for each $vw \in S$
3. $p_{vw} = 0$; for each $vw \notin S$
4. $\sum_{w \in E(v)} p_{vw} = 1$; for each non-stochastic vertex v
5. $p_{vw} = \mathsf{p}(vw)$; for all stochastic vertices v.

The variables r_v^i correspond to the payoff player i receives by starting from the vertex v. The payoffs r_v^i are equal to 0 for all vertices in V_S because no terminal is reachable from such vertices. The payoffs for other vertices satisfy the following constraints.

6. $r_t^i = \mu_i(t)$, for $t \in T$;
7. $r_v^i = 0$, for $v \notin V_S$;
8. $r_v^i = \sum_{w \in E(v)} p_{vw}\, r_v^i$, for $v \in V_S \backslash T$;

We now check that there is no unilateral deviation for the players by adding the following constraints. This following check is enough because, when fixing the strategies of all players except player i, the resulting structure is an MDP. For MDPs, Bellman equations characterise the value of each state and the equation below enforces that the value at a state v is at least as large as the maximum value of any successor, thereby ensuring consistency with the Bellman equations of the induced MDP [4, Theorem 10.109].

9. $r_v^i \geqslant r_w^i$, for $v \in V_i, w \in E(v)$;

Finally, we add the auxiliary variables g_{vw} to keep track of the size of p_{vw}, which satisfy the following constraint:

10. $g_{vw}\, p_{vw} = 1$, for each $vw \in S$;

Thus, our ETR statement is: $\exists p\, \exists r\, \exists g$ that satisfies constraints 1-10, completing the proof. ◁

The following result from the work of Basu, Pollack, and Roy [5] guarantees that if a system of polynomial inequalities is satisfiable, then there is a valuation satisfiable within a ball with radius double-exponentially large with respect to the input size. Before we state the result, we recall that a partition of semi-algebraic subsets of $\mathbb{R}^k$ generated by a set of polynomials $\mathcal{P}$ is the collection of all nonempty sets of the form $\{x \in \mathbb{R}^k \mid \text{sign}(p(x)) = \sigma(p) \text{ for all } p \in \mathcal{P}\}$, where $\sigma : \mathcal{P} \to \{-1, 0, 1\}$ ranges over all possible sign assignments. Intuitively, the partition

is obtained by slicing $\mathbb{R}^k$ according to which side of each polynomial in $\mathcal{P}$ a point lies on (positive, zero, or negative).

▶ **Lemma 8** ([5, Theorem 1.3.5]). *Given a set $\mathcal{P}$ of polynomials of degree d in k variables with coefficients in $\mathbb{Z}$ of bit-sizes τ, then there exists a ball of radius $2^{\tau d^{O(k)}}$ that intersects every part of the partition of semi-algebraic subsets of $\mathbb{R}^k$ generated by $\mathcal{P}$ (cells of $\mathcal{P}$).*

Note that each of the constraints in the ETR sentence is a polynomial of degree at most 2. The total number of variables is $O(m|\Pi|)$. Note that we had introduced auxiliary variables g_{vw} and required that $g_{vw}\, p_{vw} = 1$, for each $vw \in S$. Since this is also a polynomial in $\mathcal{P}$, using Lemma 8, for all $vw \in S$, we have $g_{vw} \leqslant 2^{\tau 2^{O(m|\Pi|)}}$. Consequently, we have $p_{vw} \geqslant 2^{-\tau 2^{O(m|\Pi|)}}$, which completes the proof. ◀

We recall some results used in the work of Frederiksen and Milterson [14] where they adapted arguments that are standard in the context of numerical analysis to provide better algorithms for approximating the values of concurrent reachability games.

Floating-point number representation. We define the set of floating-point numbers with precision ℓ as

$$\mathcal{F}(\ell) := \left\{ m \cdot 2^e \quad | \quad m \in \{0, \cdots, 2^\ell - 1\}, \quad e \in \mathbb{Z} \right\}.$$

The floating-point representation of an element $x = m \cdot 2^e \in \mathcal{F}(\ell)$ uses $\mathsf{bit}(m) + \mathsf{bit}(|e|)$ bits. We define the relative distance of two positive real numbers $x, \widetilde{x}$ as

$$\mathsf{rel}(x, \widetilde{x}) := \max\left\{ \frac{x}{\widetilde{x}}, \frac{\widetilde{x}}{x} \right\} - 1.$$

Intuitively, $\mathsf{rel}(x, \widetilde{x})$ describes the multiplicative distance. Observe that if the values are closer to 0, but ε apart for some small ε, their relative distance is larger than if these values are much larger and only ε apart. We say x is (ℓ, i)-close to $\widetilde{x}$ if $\mathsf{rel}(x, \widetilde{x}) \leqslant (1 - 2^{1-\ell})^{-i} - 1$, where ℓ is a positive integer and i is a non-negative integer. (ℓ, i)-closeness measures how far two numbers can differ after i steps of rounding in an ℓ-bit floating-point system. Here, ℓ controls precision (smaller gaps between representable numbers), and i allows for cumulative rounding error. So x is (ℓ, i)-close to $\widetilde{x}$ if their relative difference is within about $i \cdot 2^{1-\ell}$, that is, within i units of machine precision.

Arithmetic operations. We define $\oplus^\ell, \ominus^\ell, \otimes^\ell, \oslash^\ell$ as finite precision arithmetic operations $+, -, *, /$ respectively by truncating the result of the exact arithmetic operation to ℓ bits. We drop the superscript ℓ if context is clear.

Floating-point probability distribution representation. We denote by $\mathcal{D}(\ell)$ the set of all floating-point probability distributions with precision ℓ. A probability distribution $\mu \in \Delta([t])$ belongs to $\mathcal{D}(\ell)$ if there exists $w_1, w_2, \cdots, w_t \in \mathcal{F}(\ell)$ such that

- For all $i \in [t]$, we have $\mu(i) = \frac{w_i}{\sum_{j \in [t]} w_j}$; and
- $\sum_{j \in [t]} w_j$ and 1 are (ℓ, t)-close.

We define the relative distance rel for probability distributions as $\mathsf{rel}(\mu, \widetilde{\mu}) := \max\{\mathsf{rel}(\mu(i), \widetilde{\mu}(i)) : i \in [t]\}$. We say μ is (ℓ, i)-close to $\widetilde{\mu}$ if $\mathsf{rel}(\mu, \widetilde{\mu}) \leqslant (1 - 2^{1-\ell})^{-i} - 1$, where ℓ is a positive integer and i is a non-negative integer.

▶ **Lemma 9.** *If there exists a constrained stationary Nash equilibrium $\bar{\sigma}$, then there exists an $(32n^2 2^{-\ell})$-constrained stationary Nash equilibrium $\bar{\sigma}'$ such that for all players i and vertices $v \in V_i$, we have $\bar{\sigma}'(v) \in \mathcal{D}(\ell)$, where $\ell \geqslant 1000n^2$.*

Proof. We first recall the result in the work of Frederiksen and Milterson [14] related to approximation of probability distributions by floating-point distributions.

▶ **Lemma 10** ([14, Lemma 5])**.** *Consider $x_1, \cdots, x_t \in \mathcal{F}(\ell)$. Let $\mu(i) \coloneqq x_i \oslash \left(\bigoplus_{j=1}^{t} x_j\right)$. Then, there exists $\widetilde{\mu} \in \mathcal{D}(\ell)$ such that for all i, we have $\widetilde{\mu}(i) = \mu(i) / \left(\sum_{j=1}^{t} \mu(j)\right)$, and μ and $\widetilde{\mu}$ are $(\ell, 2t)$-close.*

A stationary strategy comprises of probability distributions over actions for each state. We can truncate a strategy by truncating each of these probability distributions. Let the strategy profile $\bar{\sigma}'$ be the truncation of the strategy profile $\bar{\sigma}$ defined in the above result. Therefore, for all players i and vertices $v \in V_i$, we have that $\bar{\sigma}(v)$ and $\bar{\sigma}'(v)$ are $(\ell, 2n)$-close. Consequently, for all vertices $v, w \in V$,

$$\begin{aligned}
\mathsf{rel}(\bar{\sigma}(v)(w), \bar{\sigma}'(v)(w)) &\leqslant \frac{1}{(1-2^{1-\ell})^{2n}} - 1 \\
&\leqslant \frac{1}{1-(2n)2^{1-\ell}} - 1 && \text{(Bernoulli inequality)} \\
&\leqslant \frac{(2n)2^{1-\ell}}{1-(2n)2^{1-\ell}} && \text{(rearrange)} \\
&\leqslant 4n2^{-\ell} && \left(\ell \geqslant 1000n^2\right) \qquad (1)
\end{aligned}$$

We then recall the result in the work of Solan [21]. This result provides an upper bound on the difference between the reachability values of two Markov chains based on the relative distance of their transition functions.

▶ **Lemma 11** ([21, Thm. 6])**.** *Consider two MCs $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ with identical vertex sets and a target set T. Let μ be a reward function, where $\mu(v) = 1$ for all vertices $v \in T$ and $\mu(v) = 0$ for all vertices $v \in V \backslash T$. We denote by val and $\widetilde{\mathsf{val}}$ the expected payoff of $\mathcal{M}$ and $\widetilde{\mathcal{M}}$ respectively. Fix $\varepsilon \coloneqq \max_{v,w \in V} \mathsf{rel}(\mathsf{p}(vw), \widetilde{\mathsf{p}}(vw))$. Then, we have*

$$|\mathsf{val} - \widetilde{\mathsf{val}}| \leqslant 4n\varepsilon\,.$$

By Equation (1) and Lemma 11, we bound the the difference of payoffs for $\bar{\sigma}$ and $\bar{\sigma}'$. For all players i, we have $|\mathbb{E}(\bar{\sigma})[\mu_i] - \mathbb{E}(\bar{\sigma}')[\mu_i]| \leqslant 16n^2 2^{-\ell}$. Since the strategy profile $\bar{\sigma}$ is a constrained Nash equilibrium, $\bar{\sigma}'$ is a $(32n^2 2^{-\ell})$-constrained Nash equilibrium, which completes the proof. ◀

In the above result, we show that it is sufficient to consider floating-point representable strategies to solve the problem. We now present the $\mathsf{NP}^{\mathsf{NP}}$ procedure. We first recall the result in the work of Frederiksen and Milterson [14] related to computing the approximate value of MCs with floating-point distributions.

▶ **Lemma 12** ([14, Thm. 4])**.** *Consider an MC $\mathcal{M}$ and a target set T. Let μ be a reward function, where $\mu(v) = 1$ for all vertices $v \in T$ and $\mu(v) = 0$ for all vertices $v \in V \backslash T$. For all vertices $v \in V$, we have $\mathsf{p}(v, .) \in \mathcal{D}(\ell)$ where $\ell \geqslant 1000n^2$. Then, there exists a polynomial-time algorithm that for all vertices $v \in V$, computes an approximation $r \in \mathcal{F}(\ell)$ for $\mathbb{E}[\mu]$ such that*

$$|r - \mathbb{E}[\mu]| \leqslant 80n^4 2^{-\ell}\,,$$

where $\mathsf{val}_T(v)$ is the reachability value for the vertex v.

▶ **Lemma 13.** *The problem of deciding if the payoff for MDPs is below a threshold up to an additive error is in* coNP *where the input is a MDP $\mathcal{P}$, a reward function μ, a vertex v, a threshold $0 \leqslant \alpha \leqslant 1$, an additive error $\varepsilon = 2^{-\kappa}$ and a positive integer ℓ such that, for all stochastic vertices $w \in V$, we have*

$$\mathsf{p}(w, .) \in D(\ell), \quad \ell \geqslant 1000n^2 + \kappa\,.$$

Define $\mathsf{val} := \sup_\sigma \mathbb{E}(\sigma)[\mu]$. *Note that the numbers α and ε are represented in fixed-point binary and the NP procedure is such that*

- *If $\alpha \leqslant \mathsf{val} - \varepsilon$, then it outputs* no*; and*
- *If $\alpha \geqslant \mathsf{val} + \varepsilon$, then it outputs* yes*.*

Proof. We first present the coNP procedure and then prove its soundness and completeness.

Procedure. The procedure guesses a pure stationary strategy σ for the player. Note that the size of the representation of a pure stationary strategy is polynomial with respect to the size of the representation of $\mathcal{P}$. By fixing σ, we obtain an MC $\mathcal{M}$. We denote by val_σ the value $\mathbb{E}(\sigma)[\mu]$. By Lemma 12, there exists a polynomial time algorithm that computes an ε-approximation $\widehat{\mathsf{val}}_\sigma$ of val_σ. Our procedure outputs *no* if $\alpha \leqslant \widehat{\mathsf{val}}_\sigma$. If there exists no such pure stationary strategy, the procedure outputs *yes*.

Completeness. If $\alpha \leqslant \mathsf{val} - \varepsilon$, then, by [12], there exists a pure stationary strategy σ such that $\mathsf{val} = \mathsf{val}_\sigma$. The procedure non-deterministically guesses σ. By Lemma 12, we have $\widehat{\mathsf{val}}_\sigma + \varepsilon \geqslant \mathsf{val}_\sigma$. Therefore, we have $\alpha \leqslant \widehat{\mathsf{val}}_\sigma$, and the procedure outputs *no*.

Soundness. If $\alpha \geqslant \mathsf{val} + \varepsilon$, then for all pure stationary strategies σ, we have $\alpha \geqslant \mathsf{val}_\sigma + \varepsilon$. By Lemma 12, we have $\widehat{\mathsf{val}}_\sigma - \varepsilon \leqslant \mathsf{val}_\sigma$. Therefore, $\alpha \geqslant \widehat{\mathsf{val}}_\sigma$ which implies that the procedure outputs *yes* and yields the result. ◀

Proof of Theorem 5. We first present the procedure and then prove its soundness and completeness.

Procedure. Let ℓ be sufficiently large. The procedure guesses a strategy profile and verifies that it is an $\varepsilon/8$-constrained stationary Nash equilibrium $\bar{\sigma}$. By Lemma 6 and Lemma 9, the size of the representation of $\bar{\sigma}$ is polynomial with respect to the size of game and $\mathsf{bit}(\varepsilon)$. By fixing the strategy profile $\bar{\sigma}$, we obtain an MC $\mathcal{M}$. We denote by val_i the payoff of $\mathcal{M}$ where the reward function is μ_i. By Lemma 11, the procedure computes the $\varepsilon/8$-approximate payoff $\widehat{\mathsf{val}}_i$ in polynomial time. For each player i, by fixing the strategy profile $\bar{\sigma}_{-i}$, we obtain an MDP $\mathcal{P}_i$. We denote by $\widetilde{val}_i$ the value of $\mathcal{P}_i$ where the reward function is μ_i. Then, the procedure checks if there exists an ε-unilateral deviation for player i by deciding if $\widetilde{\mathsf{val}}_i$ for is at most $\widehat{\mathsf{val}}_i(v) + 3/4\varepsilon$ up to additive error $\varepsilon/8$. The procedure finally checks if the payoff constraints are satisfied.

Soundness. We assume that $\bar{\sigma}$ is not an ε-constrained Nash equilibrium, i.e., there exists a player i such that ε-unilateral deviation is possible. Equivalently, we have $\mathsf{val}_i \leqslant \widetilde{\mathsf{val}}_i - \varepsilon$. Therefore, we have

$$\begin{aligned} \widehat{\mathsf{val}}_i + 3/4\varepsilon &\leqslant \mathsf{val}_i + 7/8\varepsilon && \left(\widehat{\mathsf{val}}_i \text{ is an } \varepsilon/8\text{-approximation of } \mathsf{val}_i\right) \\ &\leqslant \widetilde{\mathsf{val}}_i - \varepsilon/8\,. && (\varepsilon\text{-unilateral deviation}) \end{aligned}$$

However, for this case, the coNP procedure defined in Lemma 13, successfully outputs *no*. Therefore, our procedure does not output $\bar{\sigma}$ as ε-constrained stationary Nash equilibrium, which yields the soundness of our procedure.

Completeness. By Lemma 6 and Lemma 9, there exists an $\varepsilon/8$-constrained stationary Nash equilibrium $\bar{\sigma}$ which is polynomial-size representable. The procedure non-deterministically guesses $\bar{\sigma}$. Since the strategy profile is $\varepsilon/8$-Nash equilibrium, for all players i, we have $\widetilde{\mathsf{val}}_i \leqslant \mathsf{val}_i + \varepsilon/8$. Therefore, we get

$$\begin{aligned} \widehat{\mathsf{val}}_i + 3/4\varepsilon &\geqslant \mathsf{val}_i + 5/8\varepsilon && \left(\widehat{\mathsf{val}}_i \text{ is an } \varepsilon/8\text{-approximation of } \mathsf{val}_i\right) \\ &\geqslant \widetilde{\mathsf{val}}_i + 4/8\varepsilon && (\widetilde{\mathsf{val}}_i \leqslant \mathsf{val}_i + \varepsilon/8) \\ &\geqslant \widetilde{\mathsf{val}}_i + \varepsilon/8\,. \end{aligned}$$

Thus, the coNP procedure outputs *yes*. Therefore, our procedure successfully decides that $\bar{\sigma}$ is a ε-constrained stationary Nash equilibrium. This yields the completeness of the procedure and completes the proof. ◀

4 About (very) small probabilities

We now provide a tight lower bound for Lemma 6, by showing that in a constrained NE, players might need to use double-exponentially small probabilities. This result itself is not surprising due to the $\exists\mathbb{R}$-completeness of the problem [16] even in games with 7 players or more. But we tighten this result by producing an explicit game with only 5 players where such double-exponentially small probabilities are required.

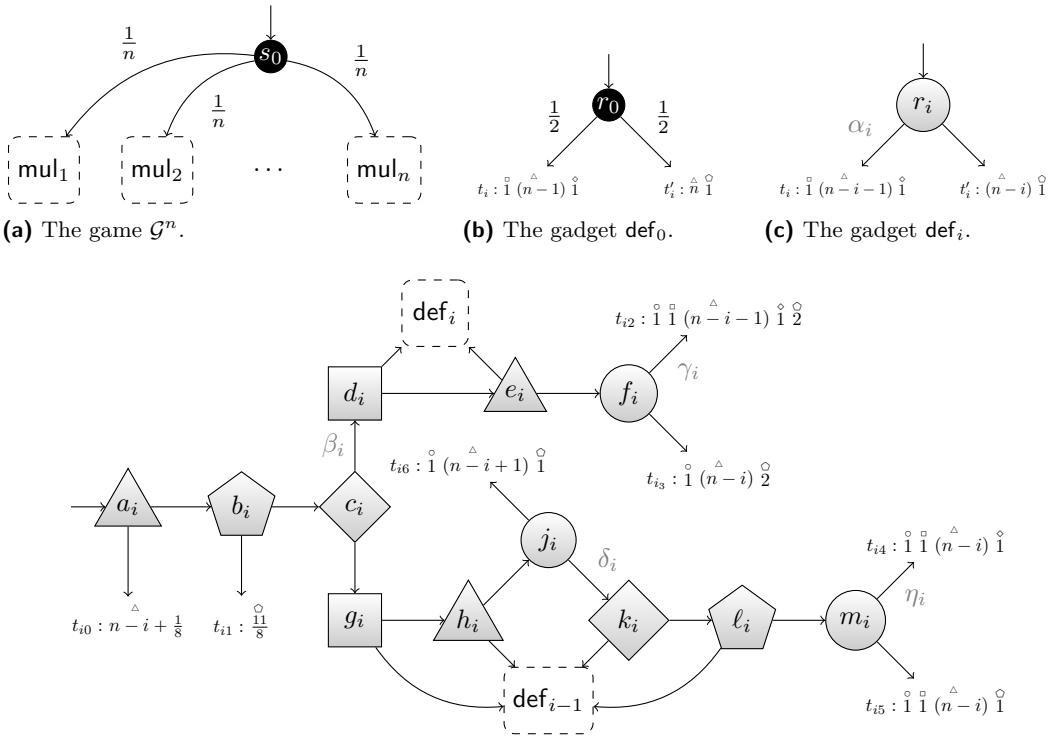
▶ **Theorem 14.** *Let $n \in \mathbb{N}$. There exist two threshold vectors $\bar{x}$ and $\bar{y}$ and a game $\mathcal{G}$ where there is a stationary Nash equilibrium $\bar{\sigma}$ satisfying $\bar{x} \leqslant \mathbb{E}(\bar{\sigma}) \leqslant \bar{y}$, but where all such stationary Nash equilibria contain probabilities that are double-exponentially small in n.*

Let $n \in \mathbb{N}$. The game $\mathcal{G}^n$ is depicted by Figure 1. It contains five players, named ○, □, △, ◇, and ⬠: the shape of each vertex indicates which player controls it, and black vertices are stochastic ones – the probability distribution is then indicated on the outgoing edges. All omitted rewards are 0. For convenience, we allow rewards greater than 1 here, accounting for the fact that all rewards can easily be normalised by dividing them by the greatest of them (namely n). For now, the reader should ignore the notations β_i, γ_i, δ_i, η_i.

Intuitively, the construction is the following: in each gadget def_i, from the vertex r_i, player ○ has to randomise between the terminal vertices t_i and t'_i (in the gadget def_0 the randomisation is imposed, because player ○ does not control the vertex r_0). The probability of going to the vertex t_i will be written α_i. We will be interested in NEs such that player ○ gets expected payoff 1: that means that the players should never go to those gadgets, nor to the terminal vertices t_{i0} or t_{i1}. But then, deviating and going to those gadgets or vertices must never be a profitable deviation. This imposes strict restrictions on which expected payoffs the players should get in each gadget mul_i, and therefore on how they can randomise their own strategies. In particular, from the vertex c_i, we will see that player ◇ must proceed to a strict randomisation, which is possible in a Nash equilibrium only if both sides are equivalent from her perspective: with this idea, each gadget mul_i binds the values of α_{i-1} and α_i, imposing $\alpha_i = \alpha_{i-1}^2$. Then, by induction, we find $\alpha_n = \frac{1}{2^{2^n}}$.

▶ **Lemma 15.** *In the game $\mathcal{G}^n$, if there exists a stationary Nash equilibrium $\bar{\sigma}$ in which player ○ gets expected payoff 1, then we have $\sigma_{\circ}(r_n) = \frac{1}{2^{2^n}}$.*

We now know that if a stationary NE satisfying this constraint exists, it necessarily includes a double-exponentially small probability. We now prove that such an NE exists.



(a) The game $\mathcal{G}^n$. **(b)** The gadget def_0. **(c)** The gadget def_i.

(d) The gadget mul_i.

Figure 1 A game where very small probabilities are necessary. The rewards corresponding to all the players not explicitly mentioned are zero.

▶ **Lemma 16.** *For every $n \in \mathbb{N}$, the game $\mathcal{G}^n$ has a stationary Nash equilibrium where player $\circ$'s expected payoff is 1.*

Together, Lemmas 15 and 16 prove Theorem 14. This theorem should be understood as a lower bound on Lemma 6, that shows that one cannot expect a better complexity than $\mathsf{NP}^{\mathsf{NP}}$ using the techniques we use here.

However, let us now note that the above result is true only for exact Nash equilibria. For ε-NEs, our construction does not lead to the same result, as shown by the following lemma.

▶ **Lemma 17.** *Let $n \in \mathbb{N}$ and $\varepsilon > 0$. In the game $\mathcal{G}^n$, there exists a stationary ε-NE $\bar{\tau}$ where player $\circ$'s expected payoff is 1, and where for each player i and every edge uv with $u \in V_i$, the size $\mathsf{bit}(\tau_i(u)(v))$ is bounded by a polynomial of $\mathsf{bit}(n)$ and $\mathsf{bit}(\varepsilon)$.*

We conjecture that this can be generalised, and that as soon as we consider ε-NEs instead of NEs, double-exponentially small probabilities are no longer required.

▶ **Conjecture 18.** *Let $\mathcal{G}$ be a game, let $\bar{x}, \bar{y}$ be two vectors denoting the constraints, and let $\varepsilon > 0$. If there exists a stationary ε-NE in $\mathcal{G}$ where the expected payoffs are between $\bar{x}$ and $\bar{y}$, then there exists one that has polynomial representation.*

Proving such a result would lead to an NP upperbound on our approximate problem, since such an ε-NE can then be guessed and checked in polynomial time.

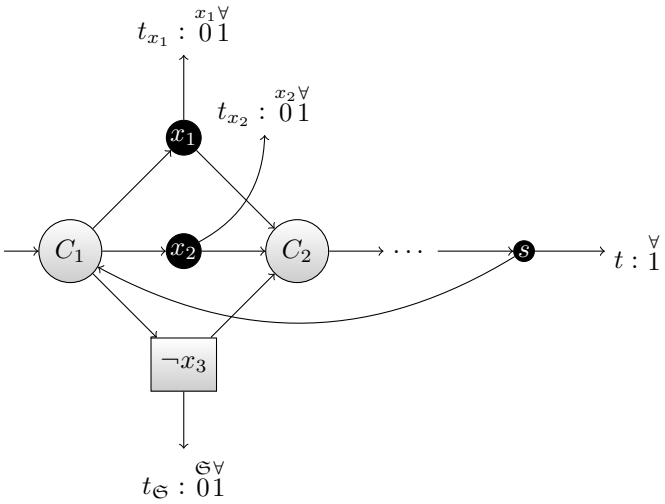


Figure 2 A reduction from 3SAT.

5 A lower bound

We dedicate this section to give a lower bound on our problem, and we show NP-hardness of the approximate version of the constrained existence problem.

▶ **Theorem 19.** *The approximate constrained existence problem of NEs is* NP*-hard.*

Proof. We proceed by reduction from the 3SAT problem. Let $\varphi = \bigvee_{i=1}^{m} (L_{i1} \wedge L_{i2} \wedge L_{i3})$ be a CNF formula, over the variable set X.

Construction of the game $\mathcal{G}^\varphi$

We define the game $\mathcal{G}^\varphi$ as follows.

- There is a player called *Solver* and written $\mathfrak{S}$, and for each variable $x \in X$, there is a player x.
- For each clause C_i, there is a vertex C_i controlled by Solver. The initial vertex is C_1.
- From each clause vertex C_i, Solver can move to the vertices (C_i, L_{i1}), (C_i, L_{i2}), and (C_i, L_{i3}).
- Each vertex of the form (C_i, x) (positive literals) is a stochastic vertex. From there, there are two edges, each taken with probability $\frac{1}{2}$: one to the terminal vertex t_x, and one to the next clause vertex C_{i+1} (or the vertex s if $i = m$).
- Each vertex of the form $(C_i, \neg x)$ is controlled by player x. From there, player x can move either to the terminal vertex $t_{\mathfrak{S}}$, or to the next clause vertex C_{i+1} (or s if $i = m$).
- The vertex s is a stochastic vertex with two outgoing edges, each with probability $\frac{1}{2}$: one to the vertex C_1, one to the terminal vertex t.
- In the terminal vertex t_i, player i receives the reward 0, and every other player receives the reward 1. In the terminal vertex t, every player receives the reward 1.

That game is depicted by Figure 2, in a case where $C_1 = x_1 \vee x_2 \vee \neg x_3$. Solver controls the circle vertices, and the square vertex is controlled by player x_3. We wrote simply L for every vertex of the form (C, L). The symbol $\forall$ should be interpreted as *"every (other) player"*. Outgoing edges of stochastic vertices all have probability $\frac{1}{2}$.

Nash equilibria from a satisfiable formula

We now state the following result, whose proof is found in the full version of the paper.

▶ **Proposition 20.** *If the formula φ is satisfiable, then there exists a Nash equilibrium in the game $\mathcal{G}^\varphi$ such that Solver's expected payoff is* 1*.*

Non-existence of ε-Nash equilibria from unsatisfiable formula

For the converse, we show this stronger result, whose proof is also in full version of the paper.

▶ **Proposition 21.** *For m large enough, if the formula φ is not satisfiable, then there is no ε-Nash equilibrium in the game $\mathcal{G}^\varphi$ such that Solver's expected payoff is greater than or equal to $1-\varepsilon$, where $\varepsilon = 2^{-3m}$.*

To conclude, given the formula φ, we define the game $\mathcal{G}^\varphi$ as above, and $\varepsilon = 2^{-3m}$ (which can be written with polynomially many bits since m is given in unary by the formula). We show that, for m large enough, these games constructed are such that the unsatisfiable instances are mapped to the negative instance of the approximate existence problem (no approximate Nash equilibria satisfies the constraints), while the satisfiable formulas, and only them, are mapped to positive instances. Therefore, our problem is NP-hard. ◀

We conclude by saying that in the full version of this paper [2] we have a proof of NP hardness even when the number of players is at most two.

6 Conclusion and discussion

We studied the (functional) approximate constrained problem of NEs. Specifically, we asked the following: assuming the existence of a stationary Nash equilibrium that satisfies given payoff constraints, can we efficiently compute a stationary strategy profile that is an ε-Nash equilibrium and satisfies the constraints up to an additive error ε?

Although our results focus on turn-based stochastic games, we note that the lower bound also holds in more general settings, including concurrent games (games on graphs where the players select actions simultaneously) and broader objectives such as limit-average (mean-payoff) objectives. For the upper bound, we remark that the existential theory of reals formulations we use for turn-based games with reachability objectives can be naturally adapted to concurrent games with mean-payoff objectives, with only a constant increase in the degree of the polynomials involved. By following similar proof techniques, one can still obtain an $\mathsf{FNP}^{\mathsf{NP}}$ upper bound for the corresponding constrained ε-NE problem in these more general settings. However, we restrict our presentation and main results to turn-based games with reachability objectives in this paper to maintain clarity.

We showed that the decision version of this problem lies in the class $\mathsf{NP}^{\mathsf{NP}}$, and the corresponding search problem belongs to $\mathsf{FNP}^{\mathsf{NP}}$. On the other hand, we established only an NP-hardness lower bound. This leaves a gap in the known complexity of the problem, and closing this gap remains an open question.

One possible direction is to prove that if an ε-Nash equilibrium exists, then there also exists one in which all probabilities are at least inverse-exponential in the size of the input, assuming the existence of a Nash equilibrium. Such a result would not contradict our example in Section 4, which shows that double-exponentially small probabilities are required in the exact version of the problem. If exponentially small probabilities suffice in the approximate setting, then a strategy profile could be represented using only polynomially many bits in

the fixed-point representation. Since such a strategy yields a polynomial certificate, and verification requires only finding the values of MDPs with fixed-point representation, this would yield an NP-completeness result for the decision version.

Alternatively, if such a bound does not hold, and there exists a counterexample where all ε-Nash equilibria satisfying the constraints require double-exponentially small probabilities, this would suggest an inherent complexity. It would also point to intrinsic limitations in our techniques, analogous to those demonstrated by Hansen, Koucký, and Miltersen [15], who showed that double-exponentially small probabilities are necessary for ε-optimal strategies in concurrent stochastic games. Their result implies that any algorithm that explicitly manipulates these probabilities must use at least exponential space in the worst case, and our results would also imply the same for our problem.

We end by recalling our conjecture in Section 4.

▶ **Conjecture 18.** *Let $\mathcal{G}$ be a game, let $\bar{x}, \bar{y}$ be two vectors denoting the constraints, and let $\varepsilon > 0$. If there exists a stationary ε-NE in $\mathcal{G}$ where the expected payoffs are between $\bar{x}$ and $\bar{y}$, then there exists one that has polynomial representation.*

References

1 Agarwal Alekh, Nan Jiang, Sham M. Kakade, and Sun Wen. *Reinforcement Learning: Theory and Algorithms*. https://rltheorybook.github.io/, 2021.

2 Ali Asadi, Léonard Brice, Krishnendu Chatterjee, and K. S. Thejaswini. ε-stationary nash equilibria in multi-player stochastic graph games, 2025. doi:10.48550/arXiv.2508.15356.

3 Ali Asadi, Krishnendu Chatterjee, Raimundo Saona, and Jakub Svoboda. Concurrent Stochastic Games with Stateful-Discounted and Parity Objectives: Complexity and Algorithms. In *FSTTCS 2024*, volume 323 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:17, 2024. doi:10.4230/LIPIcs.FSTTCS.2024.5.

4 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

5 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM (JACM)*, 43(6):1002–1045, 1996. doi:10.1145/235809.235813.

6 Sougata Bose, Rasmus Ibsen-Jensen, and Patrick Totzke. Bounded-memory strategies in partial-information games. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3661814.3662096.

7 Léonard Brice, Thomas A. Henzinger, and K. S. Thejaswini. Finding Equilibria: Simpler for Pessimists, Simplest for Optimists. In *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345, pages 30:1–30:18, 2025. doi:10.4230/LIPIcs.MFCS.2025.30.

8 Krishnendu Chatterjee, Rupak Majumdar, and Marcin Jurdzinski. On Nash equilibria in stochastic games. In *Computer Science Logic, 18th International Workshop, CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2004. doi:10.1007/978-3-540-30124-0_6.

9 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

10 Luca de Alfaro, Thomas A. Henzinger, and Ranjit Jhala. Compositional methods for probabilistic systems. In *CONCUR 2001 — Concurrency Theory*, pages 351–365, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-44685-0_24.

11 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Approximating the existential theory of the reals. In George Christodoulou and Tobias Harks, editors, *Web and Internet Economics - 14th International Conference, WINE 2018, Oxford,*

UK, December 15-17, 2018, Proceedings, volume 11316 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2018. doi:10.1007/978-3-030-04612-5_9.

12 Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

13 Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. *Automated Verification Techniques for Probabilistic Systems*, pages 53–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-21455-4_3.

14 Søren Kristoffer Stiil Frederiksen and Peter Bro Miltersen. Approximating the value of a concurrent reachability game in the polynomial time hierarchy. In *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 457–467. Springer, 2013. doi:10.1007/978-3-642-45030-3_43.

15 Kristoffer Arnsfelt Hansen, Michal Koucký, and Peter Bro Miltersen. Winning concurrent reachability games requires doubly-exponential patience. In *LICS*, pages 332–341. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.44.

16 Kristoffer Arnsfelt Hansen and Steffan Christ Sølvsten. $\exists\mathbb{R}$-completeness of stationary nash equilibria in perfect information stochastic games. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.45.

17 Claude Lefèvre. Optimal control of a birth and death epidemic process. *Operations Research*, 29(5):971–982, 1981. doi:10.1287/OPRE.29.5.971.

18 Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In Daniel A. Menascé and Noam Nisan, editors, *Proceedings 4th ACM Conference on Electronic Commerce (EC-2003), San Diego, California, USA, June 9-12, 2003*, pages 36–41. ACM, 2003. doi:10.1145/779928.779933.

19 John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. doi:10.1073/pnas.36.1.48.

20 Dawei Shi, Robert J Elliott, and Tongwen Chen. On finite-state stochastic modeling and secure estimation of cyber-physical systems. *IEEE Transactions on Automatic Control*, 62(1):65–80, 2016. doi:10.1109/TAC.2016.2541919.

21 Eilon Solan. Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability*, 16:831–845, 2003.

22 Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 1231–1238. International Foundation for Autonomous Agents and Multiagent Systems, 2010. URL: https://dl.acm.org/citation.cfm?id=1838366.

23 Michael Ummels. *Stochastic multiplayer games: theory and algorithms*. PhD thesis, RWTH Aachen, Germany, January 2010.

24 Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Stochastic Multiplayer Games. *Logical Methods in Computer Science*, Volume 7, Issue 3, September 2011. doi:10.2168/LMCS-7(3:20)2011.

Cat Herding Game Played on Infinite Trees

Rylo Ashmore
Memorial University of Newfoundland, St. John's, Canada

Sophie Pinchinat
IRISA/University of Rennes, France

Abstract

The game of Cat Herding is played on a graph between two players, the cat and the herder. The game setup consists of the cat choosing a starting vertex for their cat token. Then, both players alternate turns, beginning with the herder: they delete (any) one edge, called a cut, and the cat moves along a path to a new vertex. While this game has been studied on finite graph arenas regarding how optimally herder wins, we shift our attention to an infinite version of the game where the cat may now survive indefinitely. We show that cat winning positions in an infinite tree can be characterized by a second-order monadic statement, also amounting to having a complete infinite binary tree minor, or having uncountably many distinct rays. We take advantage of the logical characterization of cat winning positions to generalize a measure known as the cat number, to ordinals.

2012 ACM Subject Classification Theory of computation → Higher order logic; Theory of computation → Verification by model checking; Theory of computation → Automata over infinite objects; Theory of computation → Tree languages

Keywords and phrases Pursuit-evasion games, Cat Herding, Cat number, Infinite trees, Monadic Second Order Logic, Ordinals

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.10

Funding *Rylo Ashmore*: Rennes Métrople, Incoming International Mobility Grant for doctoral researchers.

1 Introduction

Pursuit-evasion games is a family of games played on graphs that allow for modeling problems arising from piratical applications, such as Robotics, Network Security, and Traffic Management to mention a few. Many results on those games exist, that mostly focus on algorithms and computational complexity of the problem. For a few examples, the quintessential pursuit-evasion game of cops and robbers is known to be EXP-complete [11], the game of firefighting is NP-hard [1], and a broader survey of complexity of pursuit-evasion games can be found [6]. Some of these games also extend naturally to infinite versions, either directly or with slight rules adjustments [15, 12, 14, 4], which are typically analyzed through a lens of structural classifications, rather than algorithmic complexity. In the case of firefighting, this gives open problems, such as whether one firefighter suffices to contain a fire on a hexagonal infinite grid [9]. Such infinite games yield natural difficulties with finding algorithmic results (what is the size of the input?). This paper introduces a novel approach to tackling these difficulties, by taking a logical approach from theoretical computer science to yield deeper insight to a particular pursuit-evasion game: the *Cat Herding* game.

The game of Cat Herding, introduced in [2], is played on a graph between two players, the *cat* and the *herder*. The game setup consists of the cat choosing a starting vertex for their cat token, which we also refer to as the cat. After this, both players alternate turns, beginning with the herder: they delete (any) one edge, called a *cut*, and the cat moves along a path to a new vertex. Note that in this context, a cut refers to a single edge cut, which may or may not disconnect the graph. Note also that the cat may not stay still. Once the

© Rylo Ashmore and Sophie Pinchinat;
licensed under Creative Commons License CC-BY 4.0

cat is on an isolated vertex and cannot move (which must happen within $|E|$ cuts), the game terminates. The objective of the herder is to minimize the number of cuts required to isolate the cat, while the objective of the cat is to maximize the number of cuts required to isolate them. We also refer to the cat having no moves available as being *captured*.

While the introductory paper [2] introduces the game and analyzes it primarily on complete graphs, in this paper we shift our attention to an infinite version of the game. Here, we are concerned with if the cat can survive indefinitely. We will later find that this is not equivalent to the cat surviving arbitrarily long, and we will also provide some analysis of that. We will restrict our attention to trees due to our techniques being essentially with tree automata. For a more complete picture of general infinite Cat Herding, we direct the reader's attention to [3]. This paper's main objective is to show that a certain monadic second order logic statement is satisfied on a given tree T if, and only if, the cat can win on that tree T. We note that [5] analyzes infinite (locally finite) graphs and their localization numbers, and their results are similarly related to ends and rays. We also show that cat-win trees and trees that satisfy our Monadic Second Order Logic (MSO) statement are also equivalent to having a complete infinite binary tree minor, or having uncountably many distinct rays. Our argument allows us to also extract an explicit herder strategy on graphs that are herder-win, and this strategy involves a notion of transfinite cat numbers. We show that there are graphs with unbounded transfinite cat number (up to the countable ordinal). Our argument also allows us to extract a decision procedure for deciding if regular trees (those with a structure representable by a regular language) are cat-win or not.

The paper is organized as follows. In Section 2 we introduce the context of our study and recall some known results on the Cat Herding game on finite arenas. Then Section 3 contains our first contribution where we make use of MSO for solving the Cat Herding game on infinite tree arenas. Our second contribution on transfinite cat numbers is developed in Section 4. We end this paper with perspectives on studying the Cat Herding on other kinds of infinite arenas than trees.

2 The Cat Herding game

2.1 Basics on graphs and trees

For standard graph definitions of degree, paths, cycles, and complete graphs, we defer to [20], as well as for other more technical notions that we briefly recall here. Given a finite (undirected) graph $G = (V(G), E(G))$, where $V(G)$ is the set of vertices and $E(G) \subseteq V(G) \times V(G)$ is the set of edges, a subgraph H is a *minor* of G when it is obtainable through a sequence of vertex and edge deletions, along with edge contractions (where the edge uv is deleted, vertices u, v are identified and their neighbourhoods are merged). Minors can also be defined for infinite graphs by relaxing the former definition: we allow any set of edges and vertices to be deleted, along with any connected set of vertices to be contracted to a point analogously, etc. We also recall that any (finite or infinite) connected graph which does not contain cycles is a *tree graph*. Formally, we define a *tree* as a structure $T = (N, r, succ)$, that arises from some tree graph $G = (N, E)$, where r is a particular vertex called the *root* and $succ \subseteq N \times N$ is given by: $(u, v) \in succ$ if $(u, v) \in E$ and u is closer to r than vertex v. For finite trees, the *height* is the length of a longest branch.

When working on trees, we may use the word "node" instead of "vertex". In the following, we will write $succ(u, v)$ instead of $(u, v) \in succ$, and we will write $\leq$ for the reflexive and transitive closure of the binary relation $succ$. Namely, $u \leq v$ means either $u = v$, or u and v are on a same path from the root r in the underlying tree graph and node u is closer to r

than node v; we say that u is an *ancestor* of v (and v is a *descendant* of u). In an infinite tree $T = (N, r, succ)$, a *ray* (infinite branch) is an infinite sequence of vertices $(v_i)_{i\in\mathbb{N}}$ such that $succ(v_i, v_{i+1})$, for all i.

2.2 Cat Herding game on finite graphs

The game of Cat Herding is played on a graph, where the cat begins by choosing a vertex to start on. Then the players take turns, with the herder permanently deleting any edge from the graph, and the cat moving along any nontrivial path to a new vertex. In a finite graph, the game ends when the cat cannot move, and the score is the number of edges deleted over the course of the game. The cat is trying to maximize this number, while the herder is trying to minimize it. We will explore the infinite case on trees, but first we recall the main results on the Cat Herding game when played on finite graphs as studied in [2].

▶ **Definition 1.** *Let $G = (V(G), E(G))$ be a graph. For each $v \in V(G)$, we define* $\mathrm{cat}(G, v)$ *to be the minimum number of cuts to terminate the game (e.g., isolate the cat) played on arena G when the cat starts at v, and we let* $\mathrm{cat}(G) = \max_v \mathrm{cat}(G, v)$.

▶ **Example 2.** Actually, it can be shown by induction on the height of binary trees[1] that the cat number of any vertex is equal to its height in vertices, counting itself. For example, consider playing on the graph B_3 (the complete binary tree of height of 3 vertices). In Figure 1, diagrammed herder cuts of edges correspond to the following play where the initial position of the cat is v_ϵ. The herder makes any move, say deleting (v_ϵ, v_0). The cat responds by moving to the root of the subtree that was not cut, say v_1. The herder may cut (v_ϵ, v_1), forcing the cat down a level again, say to v_{10}. Finally the herder cuts (v_1, v_{10}), isolating the cat in 3 cuts.

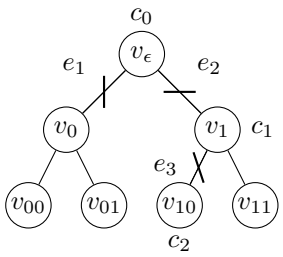


Figure 1 An optimal game on B_3. Vertex c_i denotes the cat's position on turn i, and e_i denotes the ith edge that the herder cut.

Regarding known results on the cat number, we know that it is monotonic with respect to subgraphs.

▶ **Theorem 3** ([2]). *If H is a subgraph of G, then* $\mathrm{cat}(H) \leq \mathrm{cat}(G)$. *Further, if $v \in V(H)$, then* $\mathrm{cat}(H, v) \leq \mathrm{cat}(G, v)$.

Next, we recall some preliminary values for the cat number of some known classes of graphs. As we will be interested in trees, we start with paths and stars. Let us write P_n for a path with n vertices.

[1] a tree is binary if it is non-empty and its internal nodes have at most two children.

▶ **Theorem 4** ([2]). *For all* $n \geq 2$, $\mathrm{cat}(P_n) = \lceil \log_2 n \rceil$

Paths roughly correspond to binary search, providing some intuition for the $\log_2 n$ term. After one cut, cycles are paths. Let us write C_n the cycle graph with n vertices and W_n the wheel graph with a cycle of $n-1$ vertices and an extra vertex "in the middle" connected to all of them.

▶ **Theorem 5** ([2]). *We have* $\mathrm{cat}(C_1) = 0$ *and for* $k > 1$, $\mathrm{cat}(C_{2k+1}) = \mathrm{cat}(C_{2k}) = \lceil \log_2 2k \rceil + 1$.

For $n \geq 3$, *we have* $\mathrm{cat}(W_n) = n + 1$.

Since the cat must move after a herder's cut, we have the following result on stars. Let us write S_n for the star graph with n vertices.

▶ **Theorem 6** ([2]). *For all* $n \leq 2$, $\mathrm{cat}(S_n) = 2$.

The reader interested in more details may refer to [2].

3 Cat Herding played on Infinite Trees

In this section, we classify the cat-win infinite trees. Without loss of generality, for any non-empty tree, fixing a node as the root does not change the game of Cat Herding (since the cat can travel both ways along edges of the graph).

Since our analysis will rely on monadic second-order logic, we preliminarily recall which logic we mean to use.

3.1 Monadic Second Order Logic on Trees

The Monadic Second-Order Logic (MSO) formulas we consider are meant to be interpreted on trees of arbitrary degree. Our presentation is inspired from [19].

The syntax of our monadic second-order logic (MSO) is based on a set Var composed of first-order variables $x, y, z, x', y' \ldots$ and of second-order variables $X, Y\, Z, \ldots$ which are meant to represent elements and sets, respectively. The full syntax of the logic MSO is given by :

$$\varphi, \psi := Succ(x, y) \mid X(x) \mid \varphi \wedge \psi \mid \neg\varphi \mid \exists X \colon \varphi \mid \exists x \colon \varphi$$

We also allow for the usual syntactic sugar for $\varphi \vee \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \forall x : \varphi$, and $\forall X : \varphi$. Also $x \leq y$ is a short-cut for $x = y \vee x < y$, and $X \subseteq Y$ is a short cut for $\forall x \colon X(x) \Rightarrow Y(x)$.

Further, we interpret MSO-formulas on an arbitrary tree $T = (N, r, succ)$. We will use symbols $u, v, \ldots$ for elements ranging over N and symbols $V, V', \ldots$ for subsets of N.

Given an MSO-formula φ and a valuation of the first-order and second-order variables $\nu : Var \to 2^N$ we write $T, \nu \models \varphi$ to express that formula φ holds in tree T for valuation ν, and define it by induction over φ as follows – we take the convention to write $\nu[x \mapsto v]$ for the valuation that is as ν but that maps first-order variable x onto vertex v, and similarly for $\nu[X \mapsto V]$.

- $T, \nu \models Succ(x, y)$ whenever $succ(\nu(x), \nu(y))$ in T
- $T, \nu \models X(x)$ whenever $\nu(x) \in \nu(X)$
- $T, \nu \models \neg\varphi$ whenever $T, \nu \not\models \varphi$
- $T, \nu \models \varphi \wedge \psi$ whenever $T, \nu \models \varphi$ and $T, \nu \models \psi$
- $T, \nu \models \exists x \colon \varphi$ whenever there exists $v \in N$ such that $T, \nu[x \mapsto v] \models \varphi$
- $T, \nu \models \exists X \colon \varphi$ whenever there exists $V \subseteq N$ such that $T, \nu[X \mapsto V] \models \varphi$

With a very standard construction, and thanks to second-order quantification, one can define a formula written $x \leq y$, whose semantics is the reflexive and transitive closure of the binary relation *succ* (see Section 3.3).

In the following, and for succinctness of MSO-statements, we use formulas of the form $x \leq y$ with the following semantics.

- $T, \nu \models x \leq y$ whenever $\nu(x) \leq \nu(y)$ in T

Finally, we let the formula $x < y := x \leq y \wedge \neg(y \leq x)$.

As for a closed[2] MSO-formula φ, the valuation plays no role, we simply write $T \models \varphi$. Also we loosely write $T, [x \mapsto v] \models \varphi$ to set the value of free occurrences of variable x to v in φ, and similarly for second-order variables.

We end this section by recalling the major result on MSO over infinite trees, namely Rabin's Theorem, that holds for the class of infinite complete trees with a countable degree[3]. This theorem will be useful to establish that solving the Cat Herding game over an infinite countable-degree regular tree arena is decidable, as studied in the next section.

▶ **Theorem 7** ([17, 10]). *For every $d \in \mathbb{N}$, one can decide if an input MSO-formula holds in T_d, the complete tree of degree d.*

We now focus on playing the Cat Herding on infinite graphs that have a tree structure.

3.2 MSO for Cat Herding game on infinite tree arenas

We present the core MSO-formulas over trees we will need to analyze the Cat Herding game. In our explanation for each of them, the term "path" relates to a path in the tree graph underlying the tree under concern.

$$
\begin{aligned}
\texttt{HasDiPath}(X,x,y) &:= x \leq y \wedge \forall z\colon (z \leq y \wedge x \leq z \Rightarrow X(z)) \\
&\quad \text{// } y \text{ is a descendant of } x \text{ and } X \text{ contains the unique path} \\
&\quad \text{from } x \text{ to } y \\
\texttt{DiPath}(X,x,y) &:= \texttt{HasDiPath}(X,x,y) \wedge \forall Y\colon (\texttt{HasDiPath}(Y,x,y) \Rightarrow X \subseteq Y) \\
&\quad \text{// since } X \text{ is the minimal set achieving } \texttt{HasDiPath},\ X \text{ is} \\
&\quad \text{// exactely composed of the vertices between (and including).} \\
&\quad \text{// } x \text{ and } y. \\
\texttt{Path}(Z,x,y) &:= \exists X \exists Y\colon (\forall z\colon Z(z) \Leftrightarrow (X(z) \vee Y(z))) \wedge \\
&\quad (\exists z\colon \texttt{DiPath}(X,z,x) \wedge \texttt{DiPath}(Y,z,y)) \\
&\quad \text{//} Z \text{ is a set of vertices that includes a path from } x \text{ to } y. \\
\texttt{DB}(x,y,z) &:= \exists X \exists Y\colon \texttt{Path}(X,x,y) \wedge \texttt{Path}(Y,x,z) \wedge \\
&\quad (\forall x'\colon X(x') \wedge Y(x') \Rightarrow x' = x) \\
&\quad \text{// there are edge-disjoint } x,y\text{- and } x,z\text{-paths} \\
&\quad \text{//(}\texttt{DB}\text{ means "disjoint branches").}
\end{aligned}
$$

[2] that is all variables are under the scope of a quantifier.
[3] the degree of a tree is the maximum number of children of the nodes in the tree.

We finally introduce the main formula $\texttt{DBChildin}(X, x)$ whose meaning will be further investigated and that plays a central role in characterizing cat winning positions in an infinite tree.

$$\texttt{DBChildin}(X, x) := \exists y \exists z \colon x < y \wedge x < z \wedge X(y) \wedge X(z) \wedge \texttt{DB}(x, y, z) \tag{1}$$

We now establish a few lemmas for the proof of our main result (Theorem 15) of a logical characterization of all trees where the cat has a winning strategy (that is, the cat can survive indefinitely). From now on and up to Lemma 14 included, we fix a tree $T = (N, r, succ)$. For each natural number i, we define $V_i \subseteq N$ as follows.

- $V_0 := N$, and
- for each $i > 0$, $V_i := \{v \in N \mid T, [X \mapsto V_{i-1}, x \mapsto v] \models \texttt{DBChildin}(X, x)\}$.

▶ **Lemma 8.** *For every $i \geq 0$ and $v \in N$, we have $v \in V_i$ if, and only if, v is the root of a binary tree minor of T of height i.*

Proof. We proceed by induction on i. The case $i = 0$ is trivial, as a vertex itself is a binary tree minor of height 0. Let $i > 0$.

($\Leftarrow$) Suppose $v \in N$ is the root of a complete binary tree minor of height i. We must show $v \in V_i$. We proceed by observing the left and right children from v in this minor, say u_0, u_1. These nodes are both a root of complete binary tree minor of height $i - 1$. Nodes u_0 and u_1 are the good candidates to set the two existential quantified variables y and z in formula $\texttt{DBChildin}(X, x)$: indeed, they are both descendants of v with edge-disjoint paths (by construction), and by induction, we have $u_0, u_1 \in V_{i-1}$, giving us that $v \in V_i$.

($\Rightarrow$) Suppose that $v \in V_i$. There must be u_0, u_1 such that $v < u_0, v < u_1$, $u_0, u_1 \in V_{i-1}$, and there are edge-disjoint v, u_0 and v, u_1-paths. By induction, since $u_0, u_1 \in V_{i-1}$, both u_0, u_1 must be roots of complete binary tree minors of height $i - 1$. Combined with edge-disjoint v, u_0 and v, u_1-paths, v must be the root of a binary tree minor of height i. ◀

We now extend our semantics to capture what we loosely call here "transfinite iterations" of $\texttt{DBChildin}$, and that corresponds to the fix-point of a well-identified function in Lemma 12.

▶ **Definition 9.** *As earlier, let $V_0 := N$. Then, for every successor ordinal $\alpha + 1$, let $V_{\alpha+1} := \{v \in N \mid T, [X \mapsto V_\alpha, x \mapsto v] \models \texttt{DBChildin}(X, x)\}$, and for every limit ordinal $\alpha = \{\beta | \beta < \alpha\}$, let $V_\alpha := \bigcap_{\beta \in \alpha} V_\beta$.*

▶ **Lemma 10.** *For evry ordinal α, we have $v \in V_\alpha$ if, and only if, for all $\beta < \alpha$, v has edge-disjoint descendant paths to a pair of vertices in V_β.*

Proof. We proceed by transfinite induction on α.

This is vacuously true for $\alpha = 0$. If $\alpha + 1$ is a successor ordinal, then $v \in V_{\alpha+1}$ implies there are edge-disjoint descendant paths to vertices in V_α. But vertices in V_α must have descendant paths to vertices in V_β for every $\beta < \alpha$. Composing these paths, we obtain edge-disjoint paths to every $\beta < \alpha$, as well as edge-disjoint paths to a pair of vertices in V_α, thus giving every ordinal $\beta < \alpha + 1$. Conversely, if v has edge-disjoint paths to pairs of vertices in V_β for all $\beta < \alpha + 1$, then in particular v has edge-disjoint paths to a pair of vertices in V_α since $\alpha < \alpha + 1$, and thus $v \in V_{\alpha+1}$.

If α is a limit ordinal, then $v \in V_{\beta+1}$ gives the paths to vertices in V_β, for any $\beta < \alpha$. Conversely, if v has a pair of edge-disjoint paths to all $\beta < \alpha$, then it is in all V_β by induction, and thus is in the intersection, so is in V_α. ◀

With the sets V_α and the preliminary properties expressed, we can consider the following function f over the complete lattice $(2^N, \subseteq)$.

$$\begin{array}{rccl} f: & 2^N & \to & 2^N \\ & V & \mapsto & \{v \in N \,|\, T, [X \mapsto V, x \mapsto v] \models \texttt{DBChildin}(X, x)\} \end{array}$$

One can verify that the function f is $\subseteq$-monotonic. By the Knaster-Tarski Theorem [18], the function f has a greatest fix-point, that we write $\nu V.f(V)$[4].

▶ **Lemma 11.** *If $\beta \leq \alpha$, then $V_\alpha \subseteq V_\beta$.*

Proof. We show that V_α is contained within all V_β for $\beta \leq \alpha$ by transfinite induction on α.

If α is a limit ordinal or 0, this is trivial. Otherwise, we only need to show that $V_{\alpha+1} \subseteq V_\alpha$, as this implies $V_{\alpha+1} \subseteq V_\alpha \subseteq V_\beta$ for all $\beta \leq \alpha$, the same as all $\beta < \alpha + 1$. As such, we must show that $f(V_\alpha) \subseteq V_\alpha$. For $w \in f(V_\alpha)$, we know that there are two edge-disjoint descendant paths to $u, v \in V_\alpha$. Each of these must have edge-disjoint paths to vertices in V_β for all $\beta < \alpha$ by Lemma 10. By combining these paths, we obtain that $w \in V_\alpha$ as was to be desired. ◀

With no surprise, we obtain the following.

▶ **Lemma 12.** *For every $\alpha \leq |N|$, $\nu V.f(V) \subseteq V_\alpha$. Further, $\bigcap_{\alpha \leq |N|} V_\alpha = \nu V.f(V)$.*

Proof. This a fairly standard proof, one may skip reading.

Since there is no set of all ordinals, we must bound the ordinals. However, each non-stabilizing iteration of f removes at least one vertex, thus V_α need only be computed up and including to the order of N, making this a well-defined set.

We show $\nu V.f(V) \subseteq V_\alpha$ for all ordinals α by transfinite induction. Observe $\alpha = 0$ is trivial. If $\alpha + 1$ is a successor ordinal, then we observe that $w \in \nu V.f(V)$ implies w has edge-disjoint paths to u, v also in the fix-point. But then $u, v \in V_\alpha$ by induction. Thus $w \in V_{\alpha+1}$. Finally if $\alpha = \{\beta | \beta < \alpha\}$ is a limit ordinal, then $v \in \nu V.f(V)$ implies $v \in V_\beta$ for all $\beta < \alpha$ by induction. Thus v in the fix-point implies v in the limit ordinal. Now, $\bigcap_{\alpha \leq |N|} V_\alpha \supseteq \nu V.f(V)$ follows immediately from the first argument.

We now show $\bigcap_{\alpha \leq |N|} V_\alpha \subseteq \nu V.f(V)$. We show $f(\bigcap_{\alpha \leq |N|} V_\alpha) = \bigcap_{\alpha \leq |N|} V_\alpha$, so that $\bigcap_{\alpha \leq |N|} V_\alpha$ is a fix-point, and by νX being a greatest fix-point, we obtain the subset relation. If $v \in \bigcap_{\alpha \leq |N|} V_\alpha$, then v has edge-disjoint descendant path pairs to vertices in V_α for all α (using $v \in V_{\alpha+1}$). Thus $v \in f(\bigcap_{\alpha \leq |N|} V_\alpha)$. Conversely, if $v \in f(\bigcap_{\alpha \leq |N|} V_\alpha)$, then v has edge-disjoint descendant paths to $x, y \in \bigcap_{\alpha \leq |N|} V_\alpha$. Trivially $v \in V_0$. Next, $v \in V_\alpha$ implies $v \in V_{\alpha+1}$ since v has paths to $x, y \in V_\alpha$. Finally, for limit ordinal α, if $v \in V_\beta$ for all $\beta < \alpha$, then by definition $v \in V_\alpha$. Thus $v \in \bigcap_{\alpha \leq |N|} V_\alpha$, and $\bigcap_{\alpha \leq |N|} V_\alpha$ is a fix-point of f. ◀

We will show that property $\nu V.f(V) = \emptyset$ characterizes the herder-win trees, namely trees where herder has a strategy to capture the cat (whichever node the cat starts at). In service of this, we will provide herder strategies when a given rooted subtree has bounded ordinal V_α vertices. To do so, we will make frequent use of a herder strategy to force the cat off of a ray.

▶ **Lemma 13.** *If R is a ray in tree T, then within finitely many moves, the herder may capture the cat, or force the cat off of R.*

[4] a classical notation, as for mu-calculus [13].

Proof. If the cat is not initially on R, then the herder disconnects R from the cat. On a cat move to x along this ray, the herder deletes the next vertex along the ray, isolating the cat to finitely many vertices of R. After finitely many moves, the herder may separate all of these vertices, and then make a passing move if necessary to force the cat off of the last one the cat has access to, and finally make a cut separating the cat from that vertex. Observe that as only the cat's initial move may choose how many vertices of R remain accessible after the herder's cut, removing the cat's access to this ray must be attainable within ω cuts – recall that ω is the least infinite ordinal greater than all the finite ordinals. ◀

We further explain how herder captures the cat on subtrees of bounded V_α.

▶ **Lemma 14.** *Let α be an ordinal, and let $v_0 \in N$. If for all $v \geq v_0$, $v \notin V_\alpha$, then the herder can capture the cat starting at v_0.*

Proof. Regardless of α's type, the herder begins by cutting v_0 from its parent if any, otherwise herder cuts an edge outgoing v_0. If the game does not terminate here, on the cat move to $v > v_0$, we split the argument on the type of ordinal α.

If $\alpha = 0$, the property trivially holds as it cannot be that $v \notin V_0 (= V)$.

If α is a successor ordinal, then consider where descendants in $V_{\alpha-1}$ lie in the subtree below v_0. If ever two incomparable $u, u' \in V_{\alpha-1}$, then their deepest common ancestor necessarily below or equal to v_0 would be in V_α, a contradiction. Thus all elements of $V_{\alpha-1}$ are comparable, and thus lie on a single ray (or path). Apply Lemma 13 to force the cat off of all elements of $V_{\alpha-1}$, and apply the (transfinite) induction hypothesis on the resulting tree the cat is on, since all its vertices are outside $V_{\alpha-1}$.

If $\alpha = \{\beta | \beta < \alpha\}$ is a limit ordinal, then consider the set of v_0-rooted rays $\mathcal{R} = \{R | R \textit{ is a ray}, V_\beta \cap R \neq \emptyset \textit{ for all } \beta < \alpha\}$. If $\mathcal{R}$ contains at least two distinct rays, then there must be a first vertex at which they deviate, say r. By Lemma 10, r must be in V_α, a contradiction. Thus at most one ray may contain vertices in $\bigcap_{\beta<\alpha} V_\beta$. We follow Lemma 13 to remove the cat's access to this ray, and let $x \geq v_0$, be the new cat position. Now, every descendant vertex $w \geq x$ is outside some V_{β_w}, with $\beta_w < \alpha$. We show $\sup_{w \geq x}\{\beta_w\} < \alpha$.

Suppose for contradiction $\sup_{w \geq v}\{\beta_w\} = \alpha$. By Lemma 11, there must be infinitely many $\beta < \alpha$ such that $V_\beta \neq \emptyset$. Choose a representative $w \geq x$ with $w_\beta \in V_\beta$, so that w_β is as close to the root as possible. Suppose $\beta < \beta'$ have incomparable representatives $w_\beta, w_{\beta'}$. Then their least common ancestor should have been the candidate for β. Thus every representative is comparable, and they must lie on the same ray. But then we have another cat-accessible ray r with $R \cap V_\beta \neq \emptyset$ for all $\beta < \alpha$, a contradiction with the earlier herder-play. Thus $\{\beta_w | w \geq v\}$ is bounded, say by $\beta < \alpha$. We now use the transfinite induction where vertex $x \notin V_\beta$ can be used as the new root v_0 of Lemma 14.

Thus the ordinal has been reduced for the entire subtree and the cat's location may again be interpreted as a root. ◀

Note that these ray-themed arguments are reminiscent of the argument that the cat wins implies the existence of an infinite binary tree minor, but here the greatest fix-point characterization naturally leads us to reason about particular transfinite sets, yielding us a (transfinite) constructive herder strategy for capturing the cat.

We introduce the key formula $\texttt{ECW}(x)$ (for "Exists Cat Winning") that characterizes the nodes the cat can start at and win by only downwards moves in tree. We will then make use the formula $\texttt{ECW}(x)$ (see formula $\texttt{CatWin}(x)$ in Equation (3)) to describe the set of nodes the cat can start at and survive indefinitely.

$$\texttt{ECW}(x) := \exists X \colon X(x) \wedge \forall y \colon X(y) \Rightarrow \texttt{DBChildin}(X, y) \tag{2}$$

We can now state our first main theorem.

▶ **Theorem 15.** *Let T be a tree with a designated root. The following are equivalent.*

1. *The cat wins the game of Cat Herding on T.*
2. $T \models \exists x\colon \mathit{ECW}(x)$.
3. *T contains a complete infinite binary tree minor.*
4. *T has uncountably many distinct rays from the root.*

Proof. We show more directions than necessary, as some provide different clarity on the argument.

- (1) ⇒ (2). Argue by contrapositive as follows. Negating the logical statement of cat-win graphs, we obtain $T \models \forall x\colon \forall X\colon X(x) \Rightarrow \exists y\colon X(y) \wedge \neg\mathtt{DBChildin}(X, y)$. This property in particular would hold for $X \mapsto \nu V.f(V)$ (the greatest fix-point of f) if non-empty, so that we must have $T \models \exists y\colon X(y) \wedge \neg\mathtt{DBChildin}(X, y)$. The latter yields $\nu V.f(V) \not\subseteq f(\nu V.f(V))$, a contradiction for the fix-point. Therefore $\nu V.f(V) = \emptyset$. By Lemma 12, $\bigcap_{\alpha \leq |N|} V_\alpha = \emptyset$, so that that for every node $v \in N$, we have $v \notin V_{\alpha_v}$ for some α_v. Apply Lemma 14 with v_0 the root of T, to conclude that herder can capture the cat. Observe that Lemma 14 provides an explicit herder strategy for capturing the cat.
- (2) ⇒ (3). We must construct some injective mapping h from the infinite binary tree on $\{0,1\}^*$ to a subtree of T. Let $v \in N$ be a vertex such that $T, [x \mapsto v] \models \mathtt{ECW}(x)$, and let $V \subseteq N$ be such that $T, [x \mapsto v, X \mapsto V] \models X(x) \wedge \exists y\colon X(y) \Rightarrow \mathtt{DBChildin}(X, y)$. We define the mapping h such that $h(s) \in V$, every $s \in \{0,1\}^*$.
  We proceed by induction over $s \in \{0,1\}^*$.
  - $h(\epsilon) := v \in V$ – the full binary tree minor we construct is rooted at v.
  - Let node $s \in \{0,1\}^*$ be such that $h(s)$ has already been defined and therefore $h(s) \in V$. Then, we know that $T, [x \mapsto h(s), X \mapsto V] \models X(x) \wedge \exists y\colon X(y) \Rightarrow \mathtt{DBChildin}(X, y)$, which in particular entails $T, [x \mapsto h(s), X \mapsto V] \models \mathtt{DBChildin}(X, x)$. By definition of formula $\mathtt{DBChildin}(X, x)$, node $h(s)$ has descendant nodes u_0, u_1 in V on two different branches, that we naturally use to define $h(s0)$ and $h(s1)$ respectively.

  Note that by construction h is injective, as $h(s0)$ and $h(s1)$ are disjoint and that we target vertices deeper and deeper in the tree. Say a node is *h-hit* if it is in the image of h. The complete infinite binary tree minor of T is obtained by contracting all edges with at most one h-hit node.
- (3) ⇒ (4). Immediate from the uncountability of the set of all infinite binary sequences, corresponding to rays.
- (4) ⇒ (1) by contrapositive. We must show that if the herder wins, then the rays $\mathcal{R}$ are countable. Observe the ith cut may remove the cat from accessing infinitely much of a given ray R. Let $\mathcal{R}_i$ to be the set of rays which the cat could access infinitely much of before the ith cut, but only finitely much after the ith cut. We argue $\mathcal{R}_i$ is countable and $\bigcup_i \mathcal{R}_i = \mathcal{R}$. By contradiction, if there exists an ith cut that makes $\mathcal{R}_i$ uncountable, then the cat made a bad move. Suppose the cat instead went to just on the other side of the herder cut. Then we get uncountably many rays available now, but the herder's cut may change. If iterating this process to get successively better cat moves happens to move the cat's choices upwards (towards the root), it must terminate, and $\mathcal{R}_i$ is countable. If downwards (away from the root), get uncountably many rays with same prefix, for any prefix length. In the limit, obtain ray π. Then enumerate all rays starting with π, then 1 ray deviating at depth 1, then $1, 2, 1, 2, 3, \ldots$. A given ray R has $R = \pi$ or R deviates from π at depth d. But depth d deviations are countable, and our enumeration of $\mathcal{R}$ checks depth d infinitely many times, so R is counted. Thus $\mathcal{R}$ is countable, as was to be

shown. Alternately, the cat can make a strategy that never lets $\mathcal{R}_i$ be uncountable, so that $\mathcal{R}_i$ is always countable, and there are finitely many of them. Thus $\bigcup_i \mathcal{R}_i$ is countable. But $\mathcal{R} = \bigcup_i \mathcal{R}_i$ since clearly $\mathcal{R}_i \subseteq \mathcal{R}$, and if $R \in \mathcal{R}$, then since the herder won after i steps, the cat must have access to at most one vertex of R, and so the cat must have had access to infinitely many vertices of R removed at some point, say i. Thus $R \in \mathcal{R}_i$ for some i. We have thus counted the set of all rays.

- (2) $\Rightarrow$ (1). We include this direction to make the cat's winning strategy explicit. Since $T \models \exists x\colon \texttt{ECW}(x)$, we have a node $v_0 \in N$ and a set $V \subseteq N$ such that $T, [x \mapsto v_0, X \mapsto V] \models X(x) \wedge \exists y\colon X(y) \Rightarrow \texttt{DBChildin}(X, y)$. Thus $V \neq \emptyset$. The cat's strategy will be to always play to a vertex $v \in V$ and has all herder cuts incomparable or ancestors of the cat's current position. This is possible, since on any herder cut when the cat is at v and $T, [x \mapsto v, X \mapsto V] \models \texttt{DBChildin}(X, x)$. By expanding formula $\texttt{DBChildin}(X, x)$, we get $T, [x \mapsto v, X \mapsto V] \models \exists y \exists z\colon x < y \wedge x < z \wedge X(y) \wedge X(z) \wedge \texttt{DB}(x, y, z)$, so that that there are u_0, u_1 with $T, [x \mapsto v, y \mapsto u_0, z \mapsto u_1, X \mapsto V] \models \texttt{DB}(x, y, z)$. The latter implies that no single edge cut from the herder could remove the cat's path to both u_0 and u_1, and the herder only gets one cut as $u_0, u_1 > x$, and all previous cuts were ancestors or incomparable. Thus the cat can move to u_0 or u_1, and $u_0, u_1 \in V$ are both promised, giving the invariant the cat seeks. Note this also corresponds to playing on a root of a complete infinite binary tree minor, so that (3) $\Rightarrow$ (1) could have been shown instead similarly.

It follows that conditions (1)-(4) are equivalent. ◀

As a final matter of logical completeness, we provide our second main theorem (Theorem 16), by considering the following formula that characteres all nodes that the cat can start at and still win.

$$\texttt{CatWin}(x) := \exists y \exists z\colon \texttt{DB}(x, y, z) \wedge \texttt{ECW}(y) \wedge \texttt{ECW}(z) \tag{3}$$

It is clear that the formula $\texttt{ECW}(x) \Rightarrow \texttt{CatWin}(x)$ always holds, but the reciprocal does not. To see this, consider a complete binary tree where the topmost edges are subdivided once; formally, the set of nodes is $(0 + 00 + 1 + 11)(0 \cup 1)^*$. The cat starting at node 0 can win by moving to vertex 00 or 11 (depending on herder's first cut), and because either if these two nodes satisfies $\texttt{ECW}(x)$, the cat can survive indefintely. However, the node 0 the cat started from itself does not satisfy $\texttt{ECW}(x)$: indeed, any two branches from 0 both cross edge $(0, 00)$ and therefore cannot be disjoint as required by Equation (2) for $\texttt{ECW}(x)$.

▶ **Theorem 16.** *$T, [x \mapsto v_0] \models \texttt{CatWin}(x)$ if, and only if, the cat can win when starting at v_0.*

Proof. In this proof, we may abuse notation and conflate $\texttt{ECW}, \texttt{CatWin} \subseteq N$ as a set of nodes with the predicate.

($\Leftarrow$) By contrapositive, suppose $T, [x \mapsto v_0] \models \neg\texttt{CatWin}(x)$. Then $T, [x \mapsto v_0] \models \forall y \forall z\colon \texttt{DB}(x, y, z) \Rightarrow (\neg\texttt{ECW}(y) \vee \neg\texttt{ECW}(z))$. Since by assumption $v_0 \notin \texttt{CatWin}$, and because $\texttt{ECW} \subseteq \texttt{CatWin}$, $v_0 \notin \texttt{ECW}$ so that all vertices from ECW below v_0 are on a same branch. The herder's strategy is to cut this branch from v_0. The resulting tree fails condition 2 of Theorem 15, and so the herder wins now.

($\Rightarrow$) Suppose $v_0 \in \texttt{CatWin}$. Then on the cat starting at v_0, after any cut, the cat may move to a vertex satisfying ECW (using edge-disjointness of x, u and x, v-paths). Then the strategy of (2 implies 1) in Theorem 15 gives the cat a way to win once in ECW. ◀

3.3 Decidable cases for infinite tree arenas

Note that the case where the arena is some complete d-ary tree T_d is solved immediately: For the tree T_1 is an infinite rooted path that herder in one initial cut can make finite, so that T_1 is herder-win. For T_d where $d \geq 2$, there is an infinite binary tree minor and by Theorem 15, we necessarily have $T_d \models \exists x\colon \texttt{ECW}(x)$.

The setting becomes interesting for a tree arena A that is not a complete tree T_d, but that a regular set of nodes of some complete tree T_d: Suppose that the arena A is a regular subset of nodes of some tree T_d. In such a case, A is definable by some unary MSO-predicate $A(\cdot)$. Now, to decide whether the cat wins the game on A or not consists in evaluating the formula $\exists x\colon \texttt{ECW}(x)$ "inside" the A part of the tree T_α. This is classically expressed by the *relativization* of $\exists x\colon \texttt{ECW}(x)$ w.r.t. $A(\cdot)$.

Preliminary to formally defining it, we introduce convenient notations for first-order and second-order (existential) quantifiers. For every MSO-formula φ, we let:

- $\exists^{\psi(\cdot)} x\varphi := \exists x\colon (\psi(x) \wedge \varphi)$
- $\exists^{\psi(\cdot)} X\varphi := \exists X\colon ((\forall x\colon X(x) \Rightarrow \psi(x)) \wedge \varphi$

Such quantifiers require the quantified object is confined within the set defined by the unary predicate $\psi(\cdot)$. Using such quantifiers, Formally, the *relativization* of an MSO-formula φ w.r.t. a unary MSO-predicate $\psi(\cdot)$, written $\langle\varphi|\psi(\cdot)\rangle$, can be inductively defined as follows.

$$
\begin{aligned}
\langle Succ(x,y)|\psi(\cdot)\rangle &:= Succ(x,y)\\
\langle X(x)|\psi(\cdot)\rangle &:= X(x)\\
\langle \neg\varphi|\psi(\cdot)\rangle &:= \neg\langle\varphi|\psi(\cdot)\rangle\\
\langle \varphi\wedge\psi|\psi(\cdot)\rangle &:= \langle\varphi|\psi(\cdot)\rangle \wedge \langle\psi|\psi(\cdot)\rangle\\
\langle \exists x\colon \varphi|\psi(\cdot)\rangle &:= \exists^{\psi(\cdot)} x\colon \langle\varphi|\psi(\cdot)\rangle)\\
\langle \exists X\colon \varphi|\psi(\cdot)\rangle &:= \exists^{\psi(\cdot)} X\colon \langle\varphi|\psi(\cdot)\rangle)
\end{aligned}
$$

Notice that by the propagation of the formula $\psi(\cdot)$ down the syntactic tree of the original formula φ, the quantifier alternation depth (q.a.d for short) of the formula $\langle\varphi|\psi(\cdot)\rangle$ is at most the sum of the q.a.d of each formula, plus one since the outermost quantifier of the propagated formula might occur in the scope of a dual quantifier in the original formula.

We can now formally define the Cat Herding Problem we consider.

Cat Herding Problem

Input:	A degree $d \in \mathbb{N}$ and an MSO-formula $A(\cdot)$.
Output:	$T_d \models \langle(\exists x\colon \texttt{ECW}(x))|A(\cdot)\rangle$?

By Rabin's Theorem [17] we know that the Cat Herding Problem is decidable. We below provide an upper bound for it.

Note that when resorting to alternating automata constructions [16], model checking an MSO-formula on the tree T_d requires some nested exponential-time operations that depend on its *quantifier alternation depth* (shortly written q.a.d in the remaining of the paper). The q.a.d of an MSO-formula is the maximum number of alternations between existential and universal quantifiers along any path in the formula's parse tree.

In particular, when the MSO-formula with q.a.d k has an existential outermost quantifier and is expressed only with atomic formulas based on the binary symbol *Succ*, the model checking can be done in $(k+1)$-EXPTIME. The original result comes from the logic SdS, the monadic second-order theory with d-successors, see [17].

It therefore remains to determine the q.a.d of our formula $\langle \exists x\colon \texttt{ECW}(x) | A(\cdot) \rangle$ to obtain an upper bound complexity. However, we preliminary need to translate every occurrence of atomic formulas of the form $x \leq y$ in terms of the *Succ* predicate. This is a classic translation that we recall here.

A formula $x \leq y$ holds of two nodes u and v in a tree $T = (N, r, succ)$ if v belongs to the descendants of u in T. We can characterize these descendants as the smallest set containing u and that is closed by the *succ* relation – recall that set $V \subseteq N$ is *succ*-closed whenever $\{w \in N \mid \exists v \in V, succ(v, w)\} \subseteq V$. We can express the *succ*-closeness in MSO:

$$\texttt{isSuccClosed}(X) := \forall y\colon (\exists x\colon X(x) \wedge Succ(x, y)) \Rightarrow X(y) \tag{4}$$

We now define the descendants of x, namely the smallest *succ*-closed set that contains x:

$$\begin{aligned} \texttt{isDescendantsOf}(X, x) := \quad & X(x) \wedge \texttt{isSuccClosed}(X) \wedge \\ & \forall Y\colon (Y(x) \wedge \texttt{isSuccClosed}(Y) \Rightarrow \forall y\colon (X(y) \Rightarrow Y(y))) \end{aligned} \tag{5}$$

The atomic formula $x \leq y$ now translates as follows.

$$x \leq y := \exists X(\texttt{isDescendantsOf}(X, x) \wedge X(y))$$

Note that the q.a.d of the formula $x \leq y$ is 2 (this can be seen by drawing the its parse tree) with an exitential outermost quantifier.

Noticing that formula $\exists x\colon \texttt{ECW}(x)$ as written in (2) has q.a.d 3, it can be verified that by replacing in $\exists x\colon \texttt{ECW}(x)$ every occurrence of an atomic formula of the $y \leq z$ by its translation yields a formula with a q.a.d equal to 6. Assuming that the input formula $A(\cdot)$ expressed only with predicate *Succ* (no occurrence of $\leq$) has q.a.d equal to a, we get that the translation of the main formula $\langle (\exists x\colon \texttt{ECW}(x)) | A(\cdot) \rangle$ has a q.a.d of at most $6 + a + 1$.

Because this MSO-formula has an existential outermost quantifier, the Cat Herding Problem can be solved in $(a + 8)$-EXPTIME.

4 Transfinite Cat Numbers

We introduce some transfinite structures including ordinals, which loosely correspond to the cat number, or the height of binary minors (binary minors of height α rooted at vertices in V_α). We show with a brief construction that for any given countable ordinal α, there exists an infinite tree with vertices in V_α, but still herder-win.

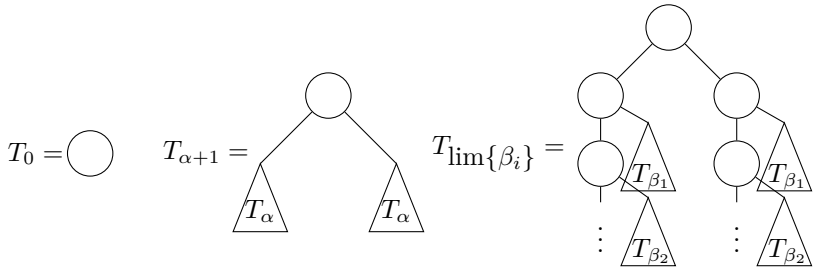


Figure 2 Binary tree constructions for every countable ordinal α.

Consider the tree constructions as depicted in Figure 2. Note that all T_α are herder-win, as every such tree allows the herder to have strategies to reduce the cat to a sub-tree $T_{\alpha'}$ with $\alpha' < \alpha$ within finitely many moves. Further, the root of every tree T_α is in V_α. This is

trivial for T_0. For successor ordinals, the root contains at least two edge-disjoint paths to vertices in $V_{\alpha-1}$ inductively. For limit ordinals $\alpha = \lim\{\beta_i\}$, every $\beta < \alpha$ has some β_i with $\beta \leq \beta_i$, and so the root having edge-disjoint paths to vertices in V_{β_i} is sufficient to show the root is in V_β for all $\beta < \alpha$, and thus the root is in V_α. Because this argument does not rely on the particular β_i sequence chosen, limit ordinals may enjoy distinct trees, but we may still conclude that the root of any tree constructed this way has a root in V_α.

Recall that the supremum of all countable ordinals is ω_1. We observe that any tree construction from countable ordinal trees with no vertices of infinite degree must be countable (by breadth first search). Thus the construction involves a countable sequence of countable ordinals, which must be countable, and thus below ω_1. The only hope for more is to allow infinite degree vertices. Indeed, once this is allowed, providing a branch from the root to tree T_β for every $\beta < \alpha$ puts the root vertex in V_α, for any limit ordinal α. This allows for the construction of any given ordinal that is still herder-win (make a passing move, cut cat onto smaller ordinal tree), but we have to give up on having binary trees, so this construction is not viable for the automatic analysis we do later.

We may also investigate whether these constructions are regular. Write $L(T_\alpha)$ for the language of addresses in tree T_α, and write $Pref(L)$ for the prefix language of L. Then, we observe that $L(T_0) = \epsilon$, $L(T_{\alpha+1}) = Pref((0 \cup 1)L(T_\alpha))$, and $L(T_{\lim\{\beta_i\}}) = Pref(0^i 1 L(T_{\beta_i}) \cup 1^i 0 L(T_{\beta_i}))$. Thus this limit ordinal construction does not yield a regular language (as there are infinitely many non-isomorphic sub-trees).

We will push an interpretation of the set of nodes V_α to a representation of the cat number. This requires generalizing our notion of cat number to ordinals. If T is an infinite tree where the cat is captured in exactly n cuts under optimal herder play, we say that $\text{cat}(T) = n$. We generalize this notion to transfinite ordinals, which to our knowledge has never been investigated in the literature.

▶ **Definition 17.** *Let $T = (N, r, succ)$ be a tree for a game of Cat Herding, $v \in N$ and α be an ordinal. We define ordinal $\text{cat}(T, v)$ as follows – we write $T_c = (N_c, r, succ)$ for the tree resulting from a single cut c in T.*

If for every cut c in T and every $\beta < \alpha$, the cat can move to a vertex $v' \in N_c$ such that $\text{cat}(T_c, v') \geq \beta$, then $\text{cat}(T, v) \geq \alpha$.

Conversely, if there exists a cut c and $\beta < \alpha$ such that all reachable vertices $v' \in N_c$ we have $\text{cat}(T_c, v') < \beta$, then $\text{cat}(T, v) < \alpha$.

We observe that the transfinite ordinal cat numbers are loosely connected to the transfinite sets V_α (the V_α are not sensitive to rays, providing a factor of error of ω).

▶ **Lemma 18.** *Let $T = (N, r, succ)$ be a tree, $v \in N$ and α be an ordinal. If $v \in V_\alpha$, then $\text{cat}(T, v) \geq \alpha$, otherwise $\text{cat}(T, v) \leq \alpha\omega$.*

Proof. If $v \in V_\alpha$, then there are edge-disjoint descendant paths to $v' \in V_\beta$ for all $\beta < \alpha$. The herder cannot block both with their edge cut, so $\text{cat}(T, v) \geq \alpha$ by Definition 17. Otherwise $v \notin V_\alpha$, then the herder's strategy is to cut above the cat – write T' for the remaining tree. If there exists $\beta < \alpha$ such that for all $v' \in V(T')$, $cat(T, v') < \beta$, then by definition $\text{cat}(T, v) < \alpha \leq \alpha\omega$. If instead for every $\beta < \alpha$ there is a reachable vertex v_β such that $\text{cat}(T', v_\beta) \geq \beta$, we have seen (proof of Lemma 14) that the set $\{v_\beta\}_{\beta<\alpha}$ of all those vertices must lie on a single ray from v. However, within finitely many moves ($< \omega$ from the ray argument in Lemma 13), the herder can force the cat to a vertex $v' \notin V_\beta$, for some $\beta < \alpha$. The resulting play involves the cat being captured in at most $\beta\omega$. Given that β may be unbounded, or $\alpha = \beta + 1$, the overall score for is at most $\beta\omega + \omega \leq \alpha\omega$. ◀

Changing the construction of Figure 2's case of successor ordinal to the same as the limit ordinal with $\beta_i = \alpha$, we obtain a new family of trees witnessing the upper bound as tight, as all vertices along the two main rays are in $V_{\alpha+1}$, so the herder must take at least ω moves (using the lower bound) to remove the cat's access to the $V_{\alpha+1}$, pushing them to the next lowest ordinal. The given construction also witnesses the lower bound, as the root of each T_α is in V_α, but no children vertices are.

▶ **Definition 19.** *Let $T = (N, r, succ)$ be a tree. We define* $\mathrm{cat}(T) = sup_{v \in N}\{\mathrm{cat}(T, v)\}$.

▶ **Corollary 20.** *If α is a countable ordinal, then there exists a herder-win binary tree T with* $\mathrm{cat}(T) \geq \alpha$. *If α is any ordinal, then there exists a (possibly locally infinite) herder-win tree T with* $\mathrm{cat}(T) \geq \alpha$.

Proof. Consider building a binary tree T_α as done in Figure 2 (we recall that for a limit ordinal α the resulting tree is sensitive to the chosen $\{\beta_i\}$ that reaches α). Still whichever T_α is obtained, we have argued that its root belongs to V_α, so that, by Lemma 18, $\mathrm{cat}(T) \geq \alpha$. ◀

5 Conclusion and Discussion

We have approached the recently introduced pursuit-evasion game Cat Herding [2] through the lens of logic, and demonstrated how logic can help in many respects.

First, the logic allows to extend the setting to particular infinite arenas, namely trees, and is amenable to finding decision procedures for the infinite class of regular infinite tree arenas. We have introduced the Cat Herding Problem and shown an upper bound for it. To our knowledge, these results are new. They pave the way to follow-up questions such as the complexity lower bounds – we currently are not aware of any concrete approach to tackle it. Also, one may consider variants where the "the cat may not stay sill" constraint is relaxed, and that do not reduce to the original problem.

Second, and interestingly, our construction involving the formula $\texttt{DBChildin}(X, x)$ motivates a notion of transfinite cat numbers, and while we provided some rough bounds on the cat numbers of vertices, our arguments exhibit a factor of ω on the bounds, leaving a natural future direction to tighten up a procedure for labeling all nodes with their cat ordinal.

Beyond immediate future direction, it remains to better understand how the logical setting could be used beyond infinite-tree arenas. In a large class of infinite graphs, such as the Caucal Hierarchy [7], our logical approach does not help to solve Cat Herding on arbitrary graphs of this hierarchy: playing Cat Herding on a given graph of the hierarchy obviously cannot reduce to playing Cat Herding in the complete binary tree – recall that in the complete binary tree, the cat can survive indefinitely, while single rays do belong to the hierarchy. Since, playing a cut in a graph of the hierarchy can be reflected as a "regular" set of cuts in the complete binary tree, there might be well-tuned new games to study on the complete binary tree that would provide the missing counterpart of playing Cat herding in those graphs. Also, it might be the case that by using the logic MSO_2 that also quantifies over edge sets, one may find a way to accurately trace back herder's cut – note that links between MSO_2 and MSO are tight for a large class of graphs [8].

References

1 Julius Althoetmar, Jamico Schade, and Torben Schürenberg. Complexity of firefighting on graphs, 2025. doi:10.48550/arXiv.2505.11082.

2 Rylo Ashmore, Danny Dyer, Trent Marbach, and Rebecca Milley. Cuts, cats, and complete graphs. *Theoretical Computer Science*, page 115393, 2025. doi:10.1016/j.tcs.2025.115393.

3 Rylo Ashmore, Danny Dyer, and Rebecca Milley. Extremal cat herding, 2025. doi:10.48550/arXiv.2505.07588.

4 Anthony Bonato, Florian Lehner, Trent Marbach, and Jd Nir. The localization game on locally finite trees. In *European Conference on Combinatorics, Graph Theory and Applications*, pages 149–155, January 2023. doi:10.5817/CZ.MUNI.EUROCOMB23-021.

5 Anthony Bonato, Florian Lehner, Trent G. Marbach, and JD Nir. Locally finite graphs and their localization numbers, 2024. doi:10.48550/arXiv.2404.02409.

6 Richard Borie, Craig Tovey, and Sven Koenig. Algorithms and complexity results for pursuit-evasion problems. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 59–66, January 2009. URL: http://ijcai.org/Proceedings/09/Papers/021.pdf.

7 D. Caucal. On infinite terms having a decidable monadic theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 165–176. Springer, 2002.

8 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

9 Alexander Dean, Sean English, Tongyun Huang, Robert A. Krueger, Andy Lee, Mose Mizrahi, and Casey Wheaton-Werle. Firefighting on the hexagonal grid and on infinite trees. *arXiv: Combinatorics*, 2020. URL: https://api.semanticscholar.org/CorpusID:222291324.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

11 William B. Kinnersley. Cops and robbers is exptime-complete. *Journal of Combinatorial Theory, Series B*, 111:201–220, 2015. doi:10.1016/j.jctb.2014.11.002.

12 Oddvar Kloster. A solution to the angel problem. *Theoretical Computer Science*, 389:152–161, December 2007. doi:10.1016/j.tcs.2007.08.006.

13 D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982. doi:10.1016/0304-3975(82)90125-6.

14 Florian Lehner. Cops, robbers, and infinite graphs, 2015. doi:10.48550/arXiv.1410.8412.

15 ANDRÁS Máthé. The angel of power 2 wins. *Combinatorics Probability and Computing.*, 16(3):363–374, May 2007. doi:10.1017/S0963548306008303.

16 D. Muller and P. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995. doi:10.1016/0304-3975(94)00214-4.

17 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. URL: http://www.jstor.org/stable/1995086.

18 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications, 1955.

19 Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1):311–346, 2002. doi:10.1016/S0304-3975(01)00185-2.

20 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, September 2000.

Stability Notions for Hospital Residents with Sizes

Haricharan Balasundaram
Indian Institute of Technology Madras, Chennai, India

J. B. Krishnashree[1] [2]
Tata Institute of Fundamental Research Mumbai, India

Girija Limaye[3]
Birla Institute of Technology and Science, Pilani, K K Birla Goa Campus, India

Meghana Nasre
Indian Institute of Technology Madras, Chennai, India

Abstract

The Hospital Residents problem with sizes (HRS) is a generalisation of the well-studied hospital residents (HR) problem. In the HRS problem, an agent a has a size $s(a)$ and the agent occupies $s(a)$ many positions of the hospital h when assigned to h. The notion of stability in this setting is suitably modified, and it is known that deciding whether an HRS instance admits a stable matching is NP-hard under severe restrictions. In this work, we explore a variation of stability, which we term occupancy-based stability. This notion was defined by McDermid and Manlove (J. of Comb. Opt. 2010) but remained unexplored to the best of our knowledge. In our work, we show that every HRS instance admits an occupancy-stable matching. We further show that computing a maximum-size occupancy-stable matching is NP-hard. We complement our hardness result by providing an approximation algorithm with a guarantee strictly better than 3 for the max-size occupancy-stable matching problem.

Given that the classical notion of stability adapted for HRS is not guaranteed to exist in general, we show a practical restriction under which a stable matching is guaranteed to exist. We present an efficient algorithm to output a stable matching in the restricted HRS instances. We also provide an alternate NP-hardness proof for the decision version of the stable matching problem for HRS which imposes a severe restriction on the number of neighbours of non-unit sized agents.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Stable matchings, Hospital Residents with sizes, Approximation algorithms, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.11

Acknowledgements We thank the anonymous reviewers for their comments that significantly improved the presentation of the paper.

1 Introduction

The Hospital Residents (HR) setting is a well-studied scenario in the literature where a set of agents, commonly called as a set of residents, needs to be assigned to hospitals. Each agent strictly orders the set of hospitals acceptable to the agent. This ordering is called the preference list. Similarly, each hospital also has a strict ordering of agents acceptable to the hospital. In addition, each hospital has a capacity that denotes the maximum number of agents that can be assigned to the hospital. The goal in the HR problem is to compute an

[1] Corresponding author.
[2] Part of this work was done during a semester-long project at IIT Madras, when the author was pursuing Integrated M.Sc., at PSG College of Technology, Coimbatore.
[3] Part of this work was done when the author was a faculty at FLAME University, Pune.


© Haricharan Balasundaram, J. B. Krishnashree, Girija Limaye, and Meghana Nasre;
licensed under Creative Commons License CC-BY 4.0

assignment of agents to hospitals which respects the capacities and is *optimal* with respect to the preferences submitted. The HR problem models several important applications, namely, recruiting medical interns (residents) in hospitals [11, 6], admitting students to degree programs; for example, the Joint Seat Allocation Authority (JOSAA) that is used in undergraduate admissions in India [8, 2], and many similar scenarios [13, 1]. In most of these settings, the de facto standard of optimality with preferences is the notion of stability. Informally, an assignment is stable if no agent hospital pair wishes to deviate from the assignment. It is well known that every instance of the HR problem admits a stable assignment [6].

Since the HR setting is relevant in various practical scenarios, the problem has been extensively studied in the literature. Furthermore, several generalisations of the HR setting are investigated. In this paper, we consider a particular variant in which agents can have sizes, and we denote this setting as the HRS setting throughout the paper. An agent a in the HRS setting has a positive size $s(a)$ associated with it. This agent can be thought of as a group of residents/students, all of whom have the same preference ordering and have to be assigned together in any assignment. Furthermore, assigning an agent a of size $s(a)$ to a hospital h occupies $s(a)$ many positions at h.

As mentioned in [5, 4], the HRS setting models multiple applications which include refugee settlement and accommodating children in day-cares. In the refugee settlement problem refugee families require multiple units of services (say, hospital beds) from each host locality. The refugee family can be represented as an agent with size proportional to the demand of the family, and the host locality with its service capacity can be represented by a hospital. A similar scenario arises in accommodating children in day-care centres. It can be realised where it is desirable to have siblings in the same day care or every child occupies a certain number of slots in the day care per day. The siblings (or the number of slots needed per child) can be represented by an agent of a given size, and day cares typically have a capacity representing how many children can be accommodated (or available slots). These correspond to hospitals in our setting.

In the presence of sizes for agents, the notion of stability is suitably adapted to take into account agent sizes (see Definition 1 and Definition 2). It is well known that an HRS instance may fail to admit a stable matching. Furthermore, deciding whether an instance of the HRS setting admits a stable matching is NP-hard [9]. Thus, the natural extension of the notion of stability does not offer a practical solution. In the work of McDermid and Manlove [9], which investigated the HRS problem, the authors had defined an alternate notion of stability, which we call occupancy-stable (see Definition 3 and Definition 4). We show that occupancy-stable matching always exists and can be computed efficiently.

As discussed in [9], the rationale behind such an extended definition is as follows: in order to accommodate a better-preferred agent, the hospital may have to displace a set of lower-preferred agents. However, this displacement may create more capacity at the hospital than what is needed for the better-preferred agent. The hospital may prefer being more occupied over getting better-preferred agents. The alternate notion of stability ensures that this requirement is fulfilled. This notion was defined in the concluding remarks in [9] but has not received attention to the best of our knowledge.

In our work, we investigate the stability notion adapted for HRS (henceforth termed as stability) as well as the notion of occupancy-stable matchings. We show that every instance of the HRS admits an occupancy-stable matching, and such a matching can be computed efficiently. We further show that the computation of maximum size occupancy-stable matching is NP-hard. To complement this hardness, we provide an efficient algorithm

with an approximation ratio strictly better than 3 for the maximum size occupancy-stable matching in an HRS instance. Furthermore, to circumvent the hardness of the decision version of the stable matching problem for HRS instances, we investigate restricted cases. Master lists in preferences is a well-studied restriction [7] where preferences of agents in a set are derived from a common ranking of the agents of the other side of the bipartition. We show that under a natural generalised master list restriction, every instance of the HRS setting admits a stable matching which can be efficiently computed.

1.1 Notation and problem definition

Let $G = (\mathcal{A} \cup \mathcal{H}, E)$ be a bipartite graph, with $n = |\mathcal{A} \cup \mathcal{H}|$ and $m = |E(G)|$. Let $\mathcal{A}$ denote the set of agents and $\mathcal{H}$ denote the set of hospitals. An edge $(a, h) \in E$ indicates that a and h are mutually acceptable. For each vertex $v \in \mathcal{A} \cup \mathcal{H}$, we let $\mathcal{N}(v)$ denote the set of acceptable partners of v, that is, the set of vertices from the other set that are adjacent to v in G. Each agent $a \in \mathcal{A}$ has an integral positive size associated with it, denoted as $s(a)$. Let $s_{\max} = \max_{a \in \mathcal{A}} s(a)$. Every hospital h has an integral positive capacity $q(h)$ associated with it, which denotes the maximum total size of agents that it can be assigned to. An agent a is called a non-unit sized agent if $s(a) > 1$.

Each agent a ranks hospitals $\mathcal{N}(a)$ in a strict order, denoted as the preference list of a. Similarly, each hospital h ranks agents in $\mathcal{N}(h)$ in a strict order, denoted as the preference list of h. For any vertex $v \in \mathcal{A} \cup \mathcal{H}$, we say that v prefers x over y iff x is before y in v's preference order, and we denote this by $x \succ_v y$ (see the example in Fig. 1, discussed later). Let ℓ_a (resp. ℓ_h) denote the length of the longest preference list of an agent (resp. hospital). We abuse the notation and call the graph $G = (\mathcal{A} \cup \mathcal{H}, E)$ along with preferences, sizes and capacities an HRS instance. It is easy to see that the HR instance [6] is a special case of HRS instance with $s_{\max} = 1$.

A many-to-one matching (called matching here onwards) in G is an assignment between agents and their acceptable hospitals such that every agent is matched to at most one hospital and for every hospital h, the sum of the sizes of agents matched to h is less than or equal to its capacity, that is, $q(h)$. Let $M(a)$ denote the hospital matched to agent a in matching M, and $M(h)$ denote the set of agents matched to hospital h in matching M. We assume that every agent prefers to be matched to one of its acceptable partners over being unmatched. The occupancy of h w.r.t. matching M is the total size of agents matched to h in M, denoted by $O_M(h)$. Thus, $O_M(h) = \sum_{a \in M(h)} s(a)$.

In an HR instance, an agent-hospital pair (a, h) is said to *block* matching M, if $(a, h) \notin M$ and they both have an incentive to deviate from M and instead, get matched to each other. That is, either a is unmatched or $h \succ_a M(a)$, and either $O_M(h) < q(h)$ or there exists an agent $a' \in M(h)$ and $a \succ_h a'$. A matching M is stable if no pair blocks it. The notion of blocking pair and stability can be extended for the HRS setting as follows [9].

▶ **Definition 1** (Blocking pair). *Given a matching M in an HRS instance G, a pair $(a, h) \in E \setminus M$ is a blocking pair w.r.t. M if $h \succ_a M(a)$ and there exists a set of agents $X \subseteq M(h)$, possibly empty, such that for every $a' \in X$, $a \succ_h a'$ and $O_M(h) - \sum_{a' \in X} s(a') + s(a) \leq q(h)$.*

The definition implies that an unmatched edge (a, h) is a blocking pair if a is either unmatched or prefers h to its current matched partner in M, and h has sufficient capacity to accommodate a by replacing a set X of agents that are currently matched to h and are lower-preferred over a. Note that if X is empty, it implies that h can accommodate a without replacing any agent(s) in $M(h)$.

(1) $a_1 : h_2 \succ h_1$ (1) $a_2 : h_1 \succ h_2$ (2) $a_3 : h_2$

[1] $h_1 : a_1 \succ a_2$ [2] $h_2 : a_2 \succ a_3 \succ a_1$

Figure 1 HRS instance that does not admit a stable matching. The number preceding the agent shows the size of the agent, whereas the number preceding the hospital denotes the hospital capacity. For example, $s(a_3) = 2$ and $q(h_2) = 2$. The preference list of an agent or a hospital is as shown. For example, hospital h_2 prefers agent a_2, followed by agent a_3, followed by agent a_1.

▶ **Definition 2** (Stable matching). *A matching M in an HRS instance G is stable if there does not exist a blocking pair w.r.t. M.*

It is well-known that an HR instance always admits a stable matching. The celebrated algorithm by Gale and Shapley efficiently computes a stable matching in an HR instance [6]. In contrast to this, an HRS instance may not admit a stable matching, as illustrated in the following example from [9]. Fig. 1 shows an example instance with $\mathcal{A} = \{a_1, a_2, a_3\}$ and $\mathcal{H} = \{h_1, h_2\}$. The preference lists of agents and hospitals are as shown in the Fig. 1. We observe that any matching that leaves a_1 unmatched cannot be stable since (a_1, h_1) blocks the matching. Similarly, any matching that leaves a_2 unmatched cannot be stable since (a_2, h_2) blocks the matching. Therefore, if a stable matching exists in the instance, it must match *both* a_1 and a_2. Irrespective of where a_1 and a_2 are matched, the capacity constraint does not allow a_3 to be matched. Thus, we consider $M = \{(a_1, h_2), (a_2, h_2)\}$, $M' = \{(a_1, h_2), (a_2, h_1)\}$ and $M'' = \{(a_1, h_1), (a_2, h_2)\}$. It is easy to verify that M is blocked by (a_2, h_1), M' is blocked by (a_3, h_2) and M'' is blocked by (a_1, h_2). Thus, the instance does not admit a stable matching.

McDermid and Manlove [9] showed that the problem of determining whether an HRS instance admits a stable matching is NP-hard. In light of this, we consider restricted settings of the instance under which the problem becomes tractable. One such restriction that has been investigated is the assumption of *master list*. The set of agents is said to have a master list ordering on them if there exists an ordering of agents such that every hospital has its preference list derived from that ordering [7]. We propose and investigate the notion of a *generalised master list* as defined below. Unlike an arbitrary HRS instance which may fail to admit a stable matching, we show that an HRS instance with the generalised master list ordering always admits a stable matching, and it can be computed efficiently.

Generalised master list on agents. We say that the preference lists of hospitals follow the generalised master list ordering on agents if there exists a partition $\{\mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \ldots, \mathcal{A}_{t_f}\}$ of $\mathcal{A}$ such that all agents in $\mathcal{A}_{t_i}$ have the same size and there exists an ordering $\langle \mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \ldots, \mathcal{A}_{t_f} \rangle$ on the sets in the partition of $\mathcal{A}$ such that every hospital $h \in \mathcal{H}$ prefers its neighbours in $\mathcal{A}_{t_i}$ over its neighbours in $\mathcal{A}_{t_j}$ iff $i < j$. We note that generalised master list ordering does *not* impose any master list ordering within a fixed set $\mathcal{A}_{t_i}$ of the partition. We remark that the example in Fig. 1 does not follow a generalised master list since there is no way to partition the three agents satisfying the required properties. Next, we illustrate generalised master lists via an example.

Let $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$ and $\mathcal{H} = \{h_1, h_2, h_3\}$. The agent preferences and the hospital capacities do not matter for the illustration and hence are left unspecified. The preferences of the hospitals, the relationship between the sizes of the agents and a partition of the agent

set are shown in Fig. 2. It is noted here that in the example, $s(a_1)$ and $s(a_4)$ could be equal. We also remark that although the generalised master list is defined on all agents in $\mathcal{A}$, the individual hospitals may have an incomplete preference list as seen in the example.

$h_1 : a_2 \succ a_1 \succ a_4$ $\quad$ $h_2 : a_1 \succ a_2 \succ a_3$ $\quad$ $h_3 : a_5 \succ a_4$

Figure 2 Illustration of generalised master lists. Assume that $s(a_1) = s(a_2) \neq s(a_3)$ and $s(a_4) = s(a_5) \neq s(a_3)$. Let $\mathcal{A} = \{\mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \mathcal{A}_{t_3}\}$ where $\mathcal{A}_{t_1} = \{a_1, a_2\}, \mathcal{A}_{t_2} = \{a_3\}, \mathcal{A}_{t_3} = \{a_4, a_5\}$.

It is easy to see that the preferences of hospitals follow the generalised master list ordering on agents because there exists an ordering $\langle \mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \mathcal{A}_{t_3} \rangle$ on the sets in the partition such that every hospital h prefers all agents in $\mathcal{A}_{t_i} \cap \mathcal{N}(h)$ over $\mathcal{A}_{t_j} \cap \mathcal{N}(h)$ if $i < j$. Since the generalised master list ordering does *not* impose any ordering within a fixed set $\mathcal{A}_{t_i}$, we note that in Fig. 2 both a_1, a_2 belong to $\mathcal{A}_{t_1}$ and yet it is acceptable that $a_2 \succ_{h_1} a_1$ and $a_1 \succ_{h_2} a_2$. We use generalised master lists in the context of the standard stability notion.

We now turn to an alternative notion of stability, which is a relaxation of the standard notion and is appealing since we show that it ensures guaranteed existence. McDermid and Manlove [9] discuss the following notion of a blocking pair, which is stronger than the one in Definition 1. We term it an occupancy-blocking pair.

▶ **Definition 3** (Occupancy-blocking pair). *Given a matching M in an HRS instance G, a pair $(a, h) \in E \setminus M$ is a blocking pair w.r.t. M if $h \succ_a M(a)$ and there exists a set of agents $X \subseteq M(h)$, possibly empty, such that $\forall a' \in X$, $a \succ_h a'$ and $O_M(h) - \sum_{a' \in X} s(a') + s(a) \leq q(h)$ and $s(a) \geq \sum_{a' \in X} s(a')$.*

That is, an edge (a, h) is an occupancy-blocking pair w.r.t. matching M if it satisfies Definition 1 and the occupancy of h does not reduce by replacing the agents in X with agent a. When a matching M does not admit any occupancy-blocking pair, then M is an occupancy-stable matching.

▶ **Definition 4** (Occupancy-stable matching). *A matching M in an HRS instance G is occupancy-stable if there does not exist an occupancy-blocking pair w.r.t. M.*

It is easy to note that every stable matching in an HRS instance is also occupancy-stable. For the HRS instance in Fig. 1 that does not admit any stable matching, consider the matching $N = \{(a_1, h_1), (a_3, h_2)\}$. The pair (a_2, h_2) is not an occupancy-blocking pair since $s(a_3) > s(a_2)$, hence N is occupancy stable. In this work, we show that every instance of the HRS problem admits an occupancy-stable matching and it can be efficiently computed.

The size of a matching in an HRS instance is the total occupancy across all hospitals. There are simple instances in which different occupancy-stable matchings have different sizes (see the example in Fig. 3). Hence, a natural question is to ask for an occupancy-stable matching which maximises the size. In this work, we show that computing such a matching is NP-hard. We complement it with a constant factor approximation algorithm.

1.2 Our results

We show the following new results for the HRS problem.

Occupancy-stable matchings. We show that every HRS instance admits an occupancy-stable matching, and such a matching can be efficiently computed. However, the problem of computing a maximum size occupancy-stable matching is NP-hard.

▶ **Theorem 5.** *An HRS instance always admits an occupancy-stable matching, and it can be computed in $O(m + n \log n)$ time.*

▶ **Theorem 6.** *Given an HRS instance, computing a maximum size occupancy-stable matching is NP-hard, even if the size of each agent and the capacity of each hospital is at most 2, and the lengths of the agents' and hospitals' preference lists are at most 4.*

On the positive side, we show that there is a strictly better than 3-factor approximation for the maximum size occupancy stable matching problem.

▶ **Theorem 7.** *Given an HRS instance, an occupancy-stable matching with an approximation guarantee of $\left(3 - \frac{1}{s_{\max}}\right)$ can be computed in $O(m + n \log n)$ time, where $s_{\max}$ denotes the maximum size of an agent.*

Stable matchings. We show that when the HRS instance admits a generalised master list ordering on agents, a stable matching always exists and it can be efficiently computed.

▶ **Theorem 8.** *Given an HRS instance with a generalised master list on agents, a stable matching always exists and it can be computed in $O(m + n)$ time.*

We note that the following are special cases of the generalised master list setting. Therefore, our algorithmic result applies to them.

- **Master list on agents**: Suppose there is a master list ordering on agents as $a'_1 \succ a'_2 \succ a'_3 \succ \ldots$. Let $\mathcal{A}_{t_i} = \{a'_i\}$. Then this is a special case of the generalised master list ordering with the ordering $\langle \mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \mathcal{A}_{t_3}, \ldots \rangle$.

Suppose that the distinct sizes of agents are $\{s_1, s_2, \ldots, s_{\max}\}$, where $s_i < s_j$ whenever $i < j$. Let $\mathcal{A}_{s_i}$ indicate the set containing all agents with size s_i.

- **All agents with larger size are preferred over all agents with smaller size:** This is a special case of the generalised master list ordering with the order $\langle \mathcal{A}_{t_1} = \mathcal{A}_{s_{\max}}, \ldots, \mathcal{A}_{t_f} = \mathcal{A}_{s_1} \rangle$
- **All agents with a smaller size are preferred over all agents with a larger size:** This is a special case of the generalised master list ordering with the order $\langle \mathcal{A}_{t_1} = \mathcal{A}_{s_1}, \ldots, \mathcal{A}_{t_f} = \mathcal{A}_{s_{\max}} \rangle$

Next, we show an alternate hardness result for the decision version of the stable matching problem in HRS instances.

▶ **Theorem 9.** *For an HRS instance, deciding whether it admits a stable matching is NP-hard even when each agent with non-unit size has exactly one hospital in its preference list.*

As noted earlier, the NP-hardness proved in [9] holds even when the agents have degree at most 3 and non-unit sized agents have size 2. In our reduction, we have non-unit sized agents with degree only 1, whereas in [9] non-unit sized agents have degree 3.

1.3 Related work

The HRS problem is related to the hospital residents problem with couples, where certain pairs of residents are identified as a couple [9]. The couples are allowed to give individual preferences as well as joint preferences. When the preferences of the couples are consistent, then the restriction is termed as Hospital residents with consistent couples. When couples need to be assigned to the same hospital, then they are called inseparable couples [9]. The HRS problem considered here is a generalisation of hospital residents with consistent and inseparable couples. Our work builds on the work done by McDermid and Manlove [9] where they considered the HRS problem. They showed that determining whether a stable matching exists in an HRS instance is NP-hard, even when each preference list has length at most three and hospital capacities are at most two.

The work of Delacretaz *et al.* [5] considered a richer model where agents can have multidimensional constraints, which is a vector of demands from multiple services provided by the hospital. In the 2019 work by Delacretaz [4], the author worked with a restricted version of the HRS problem where only two different sized agents are considered. Since deciding whether a stable matching exists is NP-hard, other fairness notions are considered. Delacretaz [4] characterised a stable matching, if one exists, using these fairness notions.

HRS setting is also relevant to job assignment settings where each job has a size or processing time. Dean *et al.* [3] studied the problem of assigning jobs to machines with preferences, generally known as the "ordinal" assignment problem. The model studied by them is the same as the HRS setting, but their feasible matching definition is relaxed. They allowed a small violation of hospital capacity. They proposed a polynomial-time algorithm that computes an unsplit (integral) job-optimal stable assignment where each machine is over-capacitated by at most the processing time of the least preferred job. In our work, we do not permit capacities to be violated, but instead consider a different notion of stability.

Master lists are motivated by real-world matching problems where the agents involved are ranked based on a common criterion. For example, as mentioned by Irving *et al.* [7], the Medical Training Application Service (MTAS) employed score-derived master lists with extensive ties for allocating medical posts. Another example is the allocation of students to dormitories at the Technion-Israel Institute of Technology [12], where master lists are used. Our generalised master lists are in a similar spirit. In the domain of solving hard stable matching problems, Meeks *et al.* [10] consider instances involving groups of similar agents, namely typed instances. In their case, the *type* of the agent determines how the agent ranks elements in the other set as well as the agent's preference list. In our case, the agent's preference list does not depend on the set in the partition that it belongs to.

2 Our algorithm

Let G be an HRS instance and let $\{\mathcal{A}_{t_1}, \ldots, \mathcal{A}_{t_f}\}$ be a partition of $\mathcal{A}$ of G such that all agents in set $\mathcal{A}_{t_i}$ have the same size. We denote such a partition of agents as a *same-sized* partition. Note that all agents in a given partition have the same size, however, it is not necessary that all agents of the same size belong to a single partition. We present an algorithm that takes as input such an instance G and an ordering on a same-sized partition of G and computes a matching. Later, we show the following two independent results:

- In Section 2.1, we show that a same-sized partition of $\mathcal{A}$ and an ordering on the sets in that partition exists such that Algorithm 1 computes an occupancy-stable matching. We further show that this matching has an approximation guarantee of strictly better than 3 with respect to the max-size occupancy-stable matching (Theorem 5 and Theorem 7).

- In Section 2.2 we assume that G admits a generalised master list ordering on agents. That is, there exists a same-sized partition of agents and an ordering on the sets of this partition. With this ordering as input, Algorithm 1 computes a stable matching (Theorem 8).

Our algorithm iterates over these sets in the given ordering. For a fixed set $\mathcal{A}_{t_k}$, it constructs an induced subgraph G_k of agents in the set $\mathcal{A}_{t_k}$ and all hospitals. We note that the graph G_k enjoys the property that all agents in the set $\mathcal{A}_{t_k}$ have the same size. Suppose the size of agents in G_k is s. The standard Gale and Shapley algorithm [6] that works with unit-sized agents can be adapted to work for uniform sized agents as follows. Agents apply in order of their preference and hospitals accept/reject based on preference. Each time an agent of size s is matched to h, the hospital h subtracts s from its available capacity instead of subtracting 1. When an agent is rejected by a hospital to accept a better preferred agent, there is no change in available capacity since all agents are of the same size. We execute this adapted version of the Gale and Shapley algorithm on G_k to compute a stable matching M_k in G_k. The edges in matching M_k computed in the k-th iteration are added to the cumulative union of stable matchings computed till then. That is, $M^{(t_k)} = \cup_{r=1}^{k} M_r$. It finally returns the result, that is, $M^{(t_f)}$. Algorithm 1 gives the pseudo-code.

Algorithm 1 Algorithm for HRS with an ordered partition of agents.

Input: an HRS instance G with an ordering $\langle \mathcal{A}_{t_1}, \ldots, \mathcal{A}_{t_f} \rangle$ on the sets in same-sized partition of $\mathcal{A}$

1: $M^{(t_0)} = \emptyset$
2: **for** $k = 1, \ldots, f$ **do**
3: **for** $h \in \mathcal{H}$ **do**
4: $q_k(h) \leftarrow q(h) - O_{M^{(t_{k-1})}}(h)$
5: let G_k denote the sub-graph of G induced by the vertices in $\mathcal{A}_{t_k} \cup \mathcal{H}$
6: let M_k be a stable matching obtained in G_k with capacities q_k by executing the adapted version of the Gale and Shapley agent-proposing algorithm
7: $M^{(t_k)} \leftarrow M^{(t_{k-1})} \cup M_k$
8: return $M^{(t_f)}$

Let $M = M^{(t_f)}$ denote the output of Algorithm 1. Before we proceed, we state important observations about our algorithm in the propositions below.

▶ **Proposition 10.** *The subgraphs G_k are pairwise edge-disjoint.*

▶ **Proposition 11.** *An edge $(a, h) \in M$ if and only if $(a, h) \in M_k$ such that $a \in \mathcal{A}_{t_k}$.*

▶ **Proposition 12.** *For every hospital h, the occupancy of h monotonically increases during the execution of the algorithm, that is, if $i \leq j$ then $O_{M^{(t_i)}}(h) \leq O_{M^{(t_j)}}(h)$.*

A stable matching in the subgraph G_k can be computed in $O(|E_k|)$ time using the adapted Gale and Shapley algorithm described above. Here E_k denotes the set of edges in the graph G_k. Since the graphs in each iteration are pairwise edge-disjoint (Proposition 10), we conclude that the algorithm runs in linear time in the size of G.

The agent-proposing adaptation of the Gale and Shapley algorithm used in line 6 of Algorithm 1 is for the sake of concreteness. We remark that in line 6 we can use any stable matching in the graph G_k and the proof of correctness in the following section will go through.

2.1 Occupancy-stable matchings

Let G be the given HRS instance. Let $s_1, s_2, \ldots, s_k$ denote the distinct sizes of agents in G and let $s_1 > s_2 > \ldots > s_k \geq 1$. Let $\mathcal{A}_{s_i}$ denote the set of agents with size s_i. It is clear that $\{\mathcal{A}_{s_1}, \ldots, \mathcal{A}_{s_k}\}$ is a same-sized partition of $\mathcal{A}$. Consider the ordering $\langle \mathcal{A}_{s_1}, \ldots, \mathcal{A}_{s_k} \rangle$ on the above partition. We note that such a same-sized partition and an ordering can be computed in $O(n \log n)$ time where $n = |\mathcal{A} \cup \mathcal{H}|$. We execute Algorithm 1 using this ordering imposed in the input instance and show that Algorithm 1 computes an occupancy-stable matching in this case. We note that the partition is only with respect to the sizes and here we do *not* have any restriction on the preferences of the hospitals with respect to this partition. Recall that we let $M = M^{(t_f)}$ denote the output of Algorithm 1.

▶ **Lemma 13.** *The matching M computed by Algorithm 1 is occupancy-stable.*

Proof. Suppose, for the sake of contradiction, that (a, h) is an occupancy-blocking pair w.r.t. M. This implies that $h \succ_a M(a)$ and one of the following two conditions must hold:

(i) $O_M(h) + s(a) \leq q(h)$ or

(ii) there exists a non-empty set of agents $X \subseteq M(h)$ such that for every $a' \in X$, $a \succ_h a'$ and $s(a) \geq \sum_{a' \in X} s(a')$ and $O_M(h) - \sum_{a' \in X} s(a') + s(a) \leq q(h)$.

We show that neither of (i) and (ii) holds, thereby leading to a contradiction to the claimed occupancy-blocking pair.

Let $s_a = s(a)$. Since (a, h) is an occupancy-blocking pair, $(a, h) \notin M$. Therefore, by Proposition 11, $(a, h) \notin M_{s_a}$. Consider the iteration when $k = s_a$. Recall that $O_{M^{(t_k)}}(h)$ denotes the occupancy of h in matching $M^{(t_k)}$, that is, $O_{M^{(t_k)}}(h)$ denotes the occupancy of h after computing the union in line 7. Since $(a, h) \notin M_{s_a}$, we have $O_{M^{(t_k)}}(h) + s(a) > q(h)$. By Proposition 12, $O_M(h) = O_{M^{(t_f)}}(h) \geq O_{M^{(t_k)}}(h)$, thus, $O_M(h) + s(a) > q(h)$ holds at the end of the algorithm. Therefore, (i) does not hold at the end of the algorithm.

Next, we show that (ii) does not hold either. We first consider the case wherein for every agent $a' \in X$, $s(a') = s_a$. Due to the stability of matching M_{s_a}, for all such agents, $a' \succ_h a$ holds. This leads to a contradiction to the given condition that for every $a' \in X$, $a \succ_h a'$. Therefore, there must exist an agent $a' \in X$ such that $s(a') \neq s(a)$.

First, suppose that there exists an agent $b \in X$ such that $s(b) > s(a)$. Then, clearly, $\sum_{a' \in X} s(a') \geq s(b) > s(a)$. This leads to a contradiction to the given condition that $s(a) \geq \sum_{a' \in X} s(a')$. Therefore, for every $a' \in X$, $s(a') < s(a)$. This implies that all such agents a' are matched to h *after* M_{s_a} and $M^{(t_k)}$ were computed.

Since $X \subseteq M$, we have $O_{M^{(t_k)}}(h) + \sum_{a' \in X} s(a') \leq O_M(h)$. Therefore, $O_M(h) - \sum_{a' \in X} s(a') \geq O_{M^{(t_k)}}(h)$. Adding $s(a)$ to both sides, we get $O_M(h) - \sum_{a' \in X} s(a') + s(a) \geq O_{M^{(t_k)}}(h) + s(a)$. As observed earlier in this proof, $O_{M^{(t_k)}}(h) + s(a) > q(h)$, therefore, $O_M(h) - \sum_{a' \in X} s(a') + s(a) > q(h)$. This leads to a contradiction to the given condition that $O_M(h) - \sum_{a' \in X} s(a') + s(a) \leq q(h)$. Therefore, (ii) does not hold. This completes the proof. ◀

Before proving the approximation guarantee, we illustrate the execution of Algorithm 1 on an example. Consider the HRS instance shown in Fig. 3. For this instance, $\mathcal{A}$ is partitioned as $\{\mathcal{A}_{t_1}, \mathcal{A}_{t_2}\}$ such that $\mathcal{A}_{t_1} = \{a_1\}, \mathcal{A}_{t_2} = \{a_2, a_3\}$ where $\mathcal{A}_{t_1}$ contains all agents of size 3 and

$\mathcal{A}_{t_2}$ contains all agents of size 2. Algorithm 1 computes $M_1 = \{(a_1, h_1)\}$ and $M_2 = \emptyset$, thereby giving the output $M = M^{(t_2)} = \{(a_1, h_1)\}$. The matching $M^* = \{(a_1, h_2), (a_2, h_1), (a_3, h_1)\}$ is the maximum size occupancy-stable matching in this instance. Thus, for this instance, we get an approximation ratio of $\frac{s(M^*)}{s(M)} = \frac{7}{3} < 3$. In what follows we prove that Algorithm 1 gives an approximation guarantee of strictly better than 3.

(3) $a_1 : h_1 \succ h_2$	(2) $a_2 : h_1$	(2) $a_3 : h_1$
[4] $h_1 : a_2 \succ a_3 \succ a_1$	[3] $h_2 : a_1$	

Figure 3 Example of HRS instance used to illustrate the execution of Algorithm 1.

Approximation guarantee. We show that the matching M computed by Algorithm 1 is a $(3 - \frac{1}{s_{\max}})$-approximation for the maximum size occupancy-stable matching. Recall that the size of a matching is the total occupancy across all hospitals. Let $s(N)$ denote the size of matching N. Then $s(N) = \sum_{h \in \mathcal{H}} O_N(h) = \sum_{a \in \mathcal{A}, N(a) \neq \perp} s(a)$.

Recall that M denotes the output of Algorithm 1.

▶ **Lemma 14.** *Let M^* be a maximum size occupancy-stable matching in G. Then $s(M^*) \leq s(M) \cdot \left(3 - \frac{1}{s_{\max}}\right)$.*

Proof. We partition the set of agents matched in M^* as follows: let X be the set of agents matched in M^* that are matched in M as well and Y be the set of agents matched in M^* but unmatched in M. That is,

- $X = \{a \mid M(a) \neq \perp \text{ and } M^*(a) \neq \perp\}$
- $Y = \{a \mid M(a) = \perp \text{ and } M^*(a) \neq \perp\}$

Clearly, $s(M^*) = \sum_{a \in X} s(a) + \sum_{a \in Y} s(a)$. We note that $\sum_{a \in X} s(a) \leq s(M)$, therefore $s(M^*) \leq s(M) + \sum_{a \in Y} s(a)$. Let $N(Y)$ denote the set of hospitals h such that for some agent $a \in Y$, $M^*(a) = h$. That is, $N(Y)$ denotes the set of matched partners in M^* of those agents who are unmatched in M. Therefore, $\sum_{a \in Y} s(a) \leq \sum_{h \in N(Y)} q(h)$.

If for every hospital $h \in N(Y)$, $q(h) \leq O_M(h) \cdot \left(2 - \frac{1}{s_{\max}}\right)$, then we get the following, thereby proving the lemma.

$$s(M^*) \leq s(M) + \sum_{a \in Y} s(a) \leq s(M) + \sum_{h \in N(Y)} q(h) \leq s(M) + \sum_{h \in N(Y)} O_M(h) \cdot \left(2 - \frac{1}{s_{\max}}\right)$$
$$\leq s(M) \cdot \left(3 - \frac{1}{s_{\max}}\right) \quad (1)$$

The last inequality follows from the fact that $\sum_{h \in N(Y)} O_M(h) \leq s(M)$. To complete the proof, it remains to show that for every $h \in N(Y)$, $q(h) \leq O_M(h) \cdot \left(2 - \frac{1}{s_{\max}}\right)$, which we do next.

By the definition of $N(Y)$, there exists at least one agent $a \in Y$ such that $M^*(a) = h$. Moreover, by the definition of Y, $M(a) = \perp$. This implies that during the execution of the algorithm, h could not accommodate a during the iteration where $M_{s(a)}$ was computed.

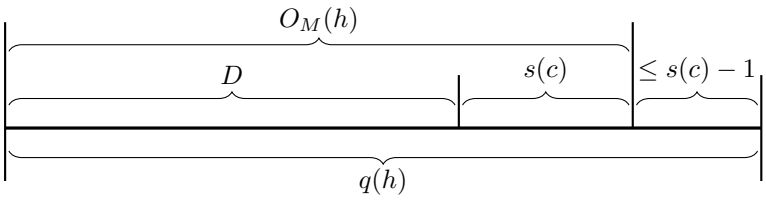


Figure 4 Relation between $O_M(h)$ and $q(h)$ in terms of D and $s(c)$.

By Proposition 12, $O_M(h) \geq O_{M^{(t_k)}}(h)$ where $a \in \mathcal{A}_{t_k}$. Since h could not occupy a in $M_{s(a)} \subseteq M^{(t_k)}$, we have $s(a) > q(h) - O_{M^{(t_k)}}(h) \geq q(h) - O_M(h)$. This implies that $q(h) < s(a) + O_M(h)$.

Next observe that at the end of the iteration that computed $M_{s(a)}$, at least one agent was matched to h. This holds, since otherwise h must have been matched to a. Moreover, the ordering imposed on the partition of the agents in input instance G implies that at that point, if $c \in M(h)$ then $s(c) \geq s(a)$. Therefore, there exists an agent $c \in M(h)$ such that $s(c) \geq s(a)$. A simple observation that $O_M(h) \geq s(c)$ gives us a 3-approximation guarantee as follows: substitute $s(a) \leq s(c) \leq O_M(h)$ in the earlier bound that we established on $q(h)$ to get $q(h) < 2 \cdot O_M(h)$. Then use this bound in Equation 1.

We refine this analysis further to achieve the claimed guarantee of $\left(3 - \frac{1}{s_{\max}}\right)$. Using $s(c) \geq s(a)$, we get $q(h) < s(c) + O_M(h) \leq s(c) - 1 + O_M(h)$. Since $c \in M(h)$, we write $O_M(h) = s(c) + D$. Therefore, $q(h) \leq D + 2 \cdot s(c) - 1$. (See Fig. 4.) Thus, we get

$$\frac{q(h)}{O_M(h)} \leq \frac{D + 2 \cdot s(c) - 1}{D + s(c)} \leq \frac{2 \cdot s(c) - 1}{s(c)} = 2 - \frac{1}{s(c)} \leq 2 - \frac{1}{s_{\max}}$$

The last inequality follows from the fact that $s(c) \leq s_{\max}$. This completes the proof. ◀

Recall that computing an ordering on the same-sized partition takes $O(n \log n)$ time, and Algorithm 1 runs in $O(m + n)$ time. These facts along with Lemma 13 establish Theorem 5. Moreover, Lemma 14 establishes Theorem 7.

2.2 Stable matchings under generalised master list

In this section, we consider HRS instances that admit a generalised master list ordering on agents. This implies that there exists a same-sized partition $\langle \mathcal{A}_{t_1}, \mathcal{A}_{t_2}, \mathcal{A}_{t_3}, \ldots \rangle$ of $\mathcal{A}$ and an ordering on it such that every hospital $h \in \mathcal{H}$ prefers its neighbours in $\mathcal{A}_{t_i}$ over its neighbours in $\mathcal{A}_{t_j}$ iff $i < j$. We show that Algorithm 1 outputs a stable matching when such an ordering is given as input.

▶ **Lemma 15.** *Suppose that G admits a generalised master list ordering on agents. When this ordering is given as input, the matching M computed by Algorithm 1 is stable.*

Proof. Assume for the sake of contradiction that there is a blocking pair (a, h) w.r.t. the matching M. Suppose that $a \in \mathcal{A}_{t_i}$ for some i. Since (a, h) blocks M, $(a, h) \notin M$. By Proposition 11, $(a, h) \notin M_{t_i}$ that was computed during the iteration when $k = i$.

Since (a, h) blocks M, there exists a set $X \subseteq M(h)$ such that for every $a' \in X$, $a \succ_h a'$ and $O_M(h) - \sum_{a' \in X} s(a') + s(a) \leq q(h)$. Let $s(X) = \sum_{a' \in X} s(a')$. Since $(a, h) \notin M_{t_i}$, we have $O_{M^{(t_i)}}(h) + s(a) > q(h)$. Moreover, due to the stability of M_{t_i} and due to the assumption that the input admits a generalised master list ordering on agents, h prefers all agents matched to it in $M^{(t_i)}$ over a. Therefore, all agents in X must be matched to h *after*

the i^{th} iteration as they are lower-preferred by h over a. To complete the proof, we need to show that even if we unmatch all agents in X, h cannot accommodate the agent a, thereby getting a contradiction to the claimed blocking pair.

By Proposition 12, $O_M(h) \geq O_{M^{(t_i)}}(h)$. In fact, since all agents in X are matched to h after $M^{(t_i)}$ was computed, we have $O_M(h) \geq O_{M^{(t_i)}}(h) + s(X)$. Thus, $O_M(h) - s(X) \geq O_{M^{(t_i)}}(h)$. Substituting $O_{M^{(t_i)}}(h) > q(h) - s(a)$, we get $O_M(h) - s(X) > q(h) - s(a)$. Hence, $O_M(h) - s(X) + s(a) > q(h)$, leading to contradiction that $O_M(h) - s(X) + s(a) \leq q(h)$. ◀

We assume that the ordering on the partitions implied by the generalised master list is part of the input instance G. This assumption, together with the fact that Algorithm 1 runs in $O(m + n)$ time establish Theorem 8.

3 NP-hardness of max-size occupancy-stable matching

In this section, we present a polynomial-time reduction to establish NP-hardness of the problem of deciding whether an $\mathcal{A}$-perfect occupancy-stable matching exists in an HRS instance, thereby proving Theorem 6. An $\mathcal{A}$-perfect occupancy-stable matching is an occupancy-stable matching that matches all agents.

The Stable Marriage problem with Incomplete lists (SMI) is the restriction of the HR problem where capacities of all hospitals are one. Conventionally, the two sets in the Stable Marriage problem are denoted as men and women instead of agents and hospitals. The Stable Marriage problem with Ties and Incomplete lists (SMTI) is a generalisation of SMI in which preference lists can have ties. The definition of a blocking pair is suitably adapted for tied instances. A pair (m, w) that is not assigned to each other in a matching M blocks M if both m and w strictly prefer each other to their current partners in M. A matching M in an SMTI instance is stable if there is no blocking pair with respect to M.

Consider a restricted instance of SMTI with an equal number of men and women, in which every woman's preference list is a strict list of length at most three; and each man's preference list is either a strict list of length exactly three or is a single tie of length two. The goal is to decide whether there exists a complete stable matching in such a restricted SMTI instance. This is the $(3, 3)$-COM-SMTI problem, which is known to be NP-hard [9].

Consider a $(3, 3)$-COM-SMTI instance I, with n men $\{m_1, m_2, \ldots, m_n\}$ and n women $\{w_1, w_2, \ldots, w_n\}$. We construct an HRS instance $G = (\mathcal{A} \cup \mathcal{H}, E)$ from I as follows. For each woman w_i we have one hospital h_i with capacity 2. For each man m_j in I with a preference list which is a tie of length two, say (w_a, w_b) where $a < b$, we create a gadget consisting of 6 agents $\{a_j^1, a_j^2, a_j^3, a_j^4, a_{j,\alpha}^1, a_{j,\alpha}^2\}$ where a_j^1 and a_j^2 have size 2 and rest have size 1; and 4 hospitals $\{h_j^1, h_j^2, h_{j,\alpha}^1, h_{j,\alpha}^2\}$ with capacities 2. See Fig. 5 (i) and (ii). For each man m_s with a strict preference list in I, we create a gadget containing two agents $a_s, a_{s,\beta}$ with size two and one respectively and one hospital $h_{s,\beta}$ with capacity 2. See Fig. 5 (iii) and (iv). This completes the description of agents and hospitals in G. Now we describe the preference lists.

- **Preference list of hospital h_i corresponding to woman w_i:** Let the preference list of w_i be l_{w_i}. We start with preference list of h_i to be set as $l_{h_i} = l_{w_i}$. For $1 \leq t \leq |l_{w_i}|$, the man m_{i_t} in l_{w_i} either has a strict preference list in I or has a single tie of length two.
  - If m_{i_t} has a strict preference list in I, then we denote the man as m_s and replace m_s in l_{h_i} by the corresponding agent a_s.
  - If m_{i_t} has a tie in its preference list, then we denote the man as m_j and denote the preference list of m_j as (w_i, w_b) in I. If $i < b$, replace m_j with a_j^1 in l_{h_i}, else with a_j^2.

  This gives us l_{h_i}, which is the preference list of hospital h_i.

(2) a_j^1 :	$h_j^1 \succ h_a \succ h_{j,\alpha}^1$	[2] h_j^1 :	$a_j^4 \succ a_j^1 \succ a_j^3$
(2) a_j^2 :	$h_j^2 \succ h_b \succ h_{j,\alpha}^2$	[2] h_j^2 :	$a_j^3 \succ a_j^2 \succ a_j^4$
(1) a_j^3 :	$h_j^1 \succ h_j^2$	[2] $h_{j,\alpha}^1$:	$a_{j,\alpha}^1 \succ a_j^1$
(1) a_j^4 :	$h_j^2 \succ h_j^1$	[2] $h_{j,\alpha}^2$:	$a_{j,\alpha}^2 \succ a_j^2$
(1) $a_{j,\alpha}^1$:	$h_{j,\alpha}^1$		
(1) $a_{j,\alpha}^2$:	$h_{j,\alpha}^2$		
	(i)		**(ii)**
(2) a_s :	$h_e \succ h_f \succ h_g \succ h_{s,\beta}$	[2] $h_{s,\beta}$:	$a_{s,\beta} \succ a_s$
(1) $a_{s,\beta}$:	$h_{s,\beta}$		
	(iii)		**(iv)**

Figure 5 Agents and hospitals corresponding to a man in the SMTI instance.

- **Preference list of vertices in the gadget corresponding to man m_j with a tie:** Recall that the preference list of m_j is a tie of length two, namely (w_a, w_b) where $a < b$. The preferences of the agents and hospitals in the gadget corresponding to m_j are shown in Fig. 5(i) and (ii). Note that the hospitals h_a and h_b corresponding to w_a and w_b appear only in the preferences of a_j^1 and a_j^2, respectively.
- **Preference list of vertices in the gadget corresponding to man m_s with strict preference list:** Let the preference list of m_s be w_e, w_f, w_g. The preference list of the agent a_s has the corresponding hospitals followed by the gadget hospital $a_{s,\beta}$. See Fig. 5(iii) and (iv) for the preferences of the vertices in the gadget corresponding to m_s.

This completes the description of the reduction. Next, we prove its correctness.

▶ **Lemma 16.** *In any occupancy-stable matching M of G, either $\{a_j^3, a_j^4\}$ are matched to the same hospital or both are unmatched.*

Proof. Assume a_j^3 is matched to h_j^1 and a_j^4 is not in the occupancy-stable matching M. Then, (a_j^1, h_j^1) blocks M, contradicting the assumption that M is $\mathcal{A}$-perfect. Say, a_j^3 is matched to h_j^2, and a_j^4 is not in an occupancy-stable matching M. Since h_j^2 has positions to accommodate a_j^4 and h_j^2 is a_j^4's first choice, (a_j^4, h_j^2) would block M. ◀

▶ **Lemma 17.** *In any $\mathcal{A}$-perfect occupancy-stable matching M of G, exactly one of $\{a_j^1, a_j^2\}$ is matched to their first choice.*

Proof. Assume both $\{a_j^1, a_j^2\}$ are matched to their first choice in M. Then, $\{a_j^3, a_j^4\}$ are unmatched in M, contradicting the assumption that M is $\mathcal{A}$-perfect. Now, assume both $\{a_j^1, a_j^2\}$ are not matched to their first choice. By Lemma 16, without loss of generality, assume both a_j^3 and a_j^4 are matched to h_j^1. Then, (a_j^2, h_j^2) blocks M, as h_j^2 has the capacity to accommodate a_j^2, contradicting the assumption that M is occupancy-stable. ◀

▶ **Lemma 18.** *In any $\mathcal{A}$-perfect occupancy-stable matching M of G, none of the agents in $\{a_j^1, a_j^2, a_s\}$ are matched to their last choice hospital.*

Proof. Since M is $\mathcal{A}$-perfect, all $\{a_j^1, a_j^2, a_s\}$ are matched to some hospital. By Lemma 17, without loss of generality, say a_j^1 is matched to h_j^1. Assume a_j^2 is matched to its last choice $h_{j,\alpha}^2$. Then $a_{j,\alpha}^2$ remains unmatched in M, contradicting the assumption that M is $\mathcal{A}$-perfect. Similarly, a_s cannot be matched to $h_{s,\beta}$. ◀

▶ **Lemma 19.** *If I admits a complete stable matching M, then G admits an $\mathcal{A}$-perfect occupancy-stable matching M'.*

Proof. Given a complete stable matching M, we describe below how to construct an $\mathcal{A}$-perfect occupancy-stable matching M' starting with an empty set. We need the following two sets to be defined.

$$\begin{aligned} T_j^a &= \{(a_j^1, h_a), (a_j^2, h_j^2), (a_j^3, h_j^1), (a_j^4, h_j^1), (a_{j,\alpha}^1, h_{j,\alpha}^1), (a_{j,\alpha}^2, h_{j,\alpha}^2)\} \\ T_j^b &= \{(a_j^1, h_j^1), (a_j^2, h_b), (a_j^3, h_j^2), (a_j^4, h_j^2), (a_{j,\alpha}^1, h_{j,\alpha}^1), (a_{j,\alpha}^2, h_{j,\alpha}^2)\} \end{aligned}$$

- For a man m_j with ties in his preference list (w_a, w_b) where $a < b$.
$$M' = M' \cup \begin{cases} T_j^a & \text{if } (m_j, w_a) \in M \\ T_j^b & \text{if } (m_j, w_b) \in M \end{cases}$$
- For a man m_s with strict preference list in I. If $(m_s, h_i) \in M$, then
$$M' = M' \cup \{(a_s, h_i), (a_{s,\beta}, h_{s,\beta})\}$$

Note that, M' is feasible and $\mathcal{A}$-perfect. To prove that M' is occupancy-stable by contradiction, assume there exists a blocking pair with respect to M'. Without loss of generality, assume $T_j^a \in M'$. Then, $a_j^2, a_j^3, a_{j,\alpha}^1, a_{j,\alpha}^2$ will not be part of a blocking pair as they are matched to their first choice. Edge (a_j^1, h_j^1) will not block M' as h_j^1 doesn't have enough capacity to accommodate a_j^1 without unmatching higher preferred agents. Edge (a_j^4, h_j^2) will not block as h_j^2 is full with higher preferred agents. If an edge (a_s, h_i) blocks M', where a_s corresponds to a man with strict preference, the respective (m_s, w_i) would be blocking M, contradicting the assumption. ◀

▶ **Lemma 20.** *If G admits an $\mathcal{A}$-perfect occupancy-stable matching M', then I admits a complete stable matching M.*

Proof. Given M', construct M as follows. Let $M = \emptyset$.

- For any j such that m_j is a man with ties in his preference list, (w_a, w_b)
$$M = M \cup \begin{cases} \{(m_j, h_a)\} & \text{if } (a_j^1, h_a) \in M' \\ \{(m_j, h_b)\} & \text{if } (a_j^2, h_b) \in M' \end{cases}$$
- For any s such that m_s is a man with strict preference list.
$$M = M \cup \{(m_s, h_i)\} \quad \text{if } (a_s, h_i) \in M'$$

By Lemma 18, exactly one of $\{a_j^1, a_j^2\}$ is matched to its top choice and other to its second choice, so there is no scenario where both $\{(m_j, h_a), (m_j, h_b)\}$ are matched. Note that the sizes of agents and capacities of hospitals are set such that every hospital that corresponds to women has almost one match. Hence, M is feasible. By Lemma 18, every man m_j that has ties is matched to either w_a or w_b. Similarly, every man m_s with a strict preference list is matched to a woman by construction. Since all men with ties in their preference list are matched, they do not block M. Suppose that there exists a man-woman pair (m_s, w_i) which blocks M where m_s is a man with a strict preference list. Then the corresponding (a_s, h_i) blocks M' in G. Therefore, M is stable in I. ◀

Note that, deciding the existence of an $\mathcal{A}$–perfect occupancy-stable matching is the same as deciding the existence of an occupancy-stable matching of size $8|M_j| + 3|M_s|$, where $|M_j|$ are $|M_s|$ are the number of men with ties in their preference list and number of men with strict preference list. Hence, Lemma 19 and Lemma 20 together establish Theorem 6.

4 NP-hardness for stable matchings in HRS

In this section, we establish the NP-hardness of the decision version of the stable matching problem. Our reduction shows that the problem remains hard even when all non-unit sized agents have degree 1. This reduction is also from $(3,3)$-COM-SMTI. Consider a $(3,3)$-COM-SMTI instance I, with n men $\{m_1, m_2, \ldots, m_n\}$ and n women $\{w_1, w_2, \ldots, w_n\}$. We construct an HRS instance $G = (\mathcal{A} \cup \mathcal{H}, E)$ from I as follows. For each woman w_i we have one hospital h_i with capacity 1. For each man m_j in I with a preference list which is a tie of length two say (w_a, w_b) where $a < b$, we create a gadget consisting of twelve agents and eight hospitals as follows. See Fig. 6(i) and (ii).

- Agents $\{a_j^1, \ldots, a_j^6\}$ where $\{a_j^1, a_j^2, a_j^3, a_j^4\}$ have unit size whereas $\{a_j^5, a_j^6\}$ have size 3.
- For each $k = 1, 2$ we have three agents $\{r_{j,t}^k \mid t = 1, 2, 3\}$ where $r_{j,1}^k, r_{j,2}^k$ have unit size and $r_{j,3}^k$ has size 3.
- Two hospitals h_j^1 and h_j^2 each with capacity 3.
- For each $k = 1, 2$, we have three hospitals $\{p_{j,t}^k \mid t = 1, 2, 3\}$ where $p_{j,3}^k$ has capacity 3 and the remaining hospitals have capacity 1.

For each man m_s with a strict preference list in I, we create a gadget containing 4 agent*s* and 3 hospitals hospital*s*. See Fig. 6 (iii) and (iv).

- A unit sized agent a_s.
- Three agents $\{r_{s,t} \mid t = 1, 2, 3\}$ where $r_{s,3}$ has size 3 and remaining agents are unit sized.
- Three hospitals $\{p_{s,t} \mid t = 1, 2, 3\}$ where $p_{s,3}$ has capacity 3 and the remaining hospitals have capacity 1.

This completes the description of the agents and hospitals in G. Now we describe the preference lists.

- **Preference list for hospital h_i corresponding to woman w_i:** Let the preference list of w_i be $l_{w_i} = m_{i_1}, m_{i_2}, m_{i_3}$. We start with preference list of h_i to be set as $l_{h_i} = l_{w_i}$. For $t = 1, 2, 3$ the man m_{i_t} in l_{w_i} either has a strict preference list in I or has a single tie of length two.
  - If m_{i_t} has a strict preference list, then we denote the man as m_s and replace m_s in l_{h_i} by the corresponding agent a_s.
  - If m_{i_t} has a tie in its preference list, then we denote the man as m_j and denote the preference list of m_j as (w_i, w_b). If $i < b$, replace m_j with a_j^1 in l_{h_i}, else with a_j^2.

  This gives us l_{h_i} which is the preference list of hospital h_i.
- **Preference lists of the vertices in the gadget corresponding to man m_j with a tie:** Recall that the preference list of m_j is a tie of length two, namely (w_a, w_b) where $a < b$. The preferences of the agents and hospitals in the gadget corresponding to m_j are shown in Fig. 6(i) and (ii). Note that the hospitals h_a and h_b corresponding to w_a and w_b appear only in the preferences of a_j^1 and a_j^2 respectively.
- **Preference lists of the vertices in the gadget corresponding to man m_s with strict preference list:** Let the preference list of m_s be w_e, w_f, w_g. Then the preference list of the agent a_s has the corresponding hospitals followed by the gadget hospital $p_{s,1}$. See Fig. 6(iii) and (iv) for the preferences of vertices in the gadget corresponding to m_s.

This completes the description of the reduction. We prove the correctness below.

(1) a_j^1 :	$h_j^1 \succ h_a \succ p_{j,1}^1$	[3] h_j^1 :	$a_j^3 \succ a_j^5 \succ a_j^1 \succ a_j^4$
(1) a_j^2 :	$h_j^2 \succ h_b \succ p_{j,1}^2$	[3] h_j^2 :	$a_j^4 \succ a_j^6 \succ a_j^2 \succ a_j^3$
(1) a_j^3 :	$h_j^2 \succ h_j^1$	[1] $p_{j,1}^k$:	$a_j^k \succ r_{j,1}^k$
(1) a_j^4 :	$h_j^1 \succ h_j^2$	[1] $p_{j,2}^k$:	$r_{j,2}^k \succ r_{j,1}^k$
(3) a_j^5 :	h_j^1	[3] $p_{j,3}^k$:	$r_{j,1}^k \succ r_{j,3}^k \succ r_{j,2}^k$
(3) a_j^6 :	h_j^2		
(1) $r_{j,1}^k$:	$p_{j,1}^k \succ p_{j,2}^k \succ p_{j,3}^k$		
(1) $r_{j,2}^k$:	$p_{j,3}^k \succ p_{j,2}^k$		
(3) $r_{j,3}^k$:	$p_{j,3}^k$		
	(i)		**(ii)**
(1) a_s :	$h_e \succ h_f \succ h_g \succ p_{s,1}$	[1] $p_{s,1}$:	$a_s \succ r_{s,1}$
(1) $r_{s,1}$:	$p_{s,1} \succ p_{s,2} \succ p_{s,3}$	[1] $p_{s,2}$:	$r_{s,2} \succ r_{s,1}$
(1) $r_{s,2}$:	$p_{s,3} \succ p_{s,2}$	[3] $p_{s,3}$:	$r_{s,1} \succ r_{s,3} \succ r_{s,2}$
(3) $r_{s,3}$:	$p_{s,3}$		
	(iii)		**(iv)**

Figure 6 Agents and hospitals corresponding to a man in the SMTI instance.

▶ **Lemma 21.** *Let m_j be a man with a tie in his preference list in I, and let $k \in \{1, 2\}$. If G admits a stable matching N, N leaves $(a_j^k, p_{j,1}^k)$ unmatched. Furthermore, N cannot leave a_j^k unmatched.*

Proof. Assume there exists j, k as above such that $(p_{j,1}^k, a_j^k) \in N$. Hence the hospital $p_{j,1}^k$ will be removed from $r_{j,1}^k$'s preference list. Now, the subgraph induced by $\{r_{j,1}^k, r_{j,2}^k, r_{j,3}^k, p_{j,2}^k, p_{j,3}^k\}$ is exactly the instance in Fig. 1 except the sizes being 3 instead of 2 and thus doesn't admit a stable matching. The second statement of the lemma follows from the fact that if a_j^k is unmatched in N then $(a_j^k, p_{j,1}^k)$ blocks N. ◀

Using similar arguments as above, we get the following lemma.

▶ **Lemma 22.** *Let m_s be a man with a strict preference list in I. Then, any matching N in G containing $(a_s, p_{s,1})$ is not stable. Furthermore, N cannot leave a_s unmatched.*

The following two lemmas (proofs in the Appendix) establish the correctness of the reduction.

▶ **Lemma 23.** *Let j be such that m_j is a man with a tie in his preference list in I. If G admits a stable matching N then one of the following holds. Either $T_j^a \subseteq N$ or $T_j^b \subseteq N$, where $T_j^a = \{(a_j^1, h_a), (a_j^2, h_j^2), (a_j^3, h_j^2), (a_j^4, h_j^2), (a_j^5, h_j^1)\}$ and $T_j^b = \{(a_j^1, h_j^1), (a_j^2, h_b), (a_j^3, h_j^1), (a_j^4, h_j^1), (a_j^6, h_j^2)\}$*

▶ **Lemma 24.** *The instance I admits a complete stable matching if and only if the reduced HRS instance G admits a stable matching.*

Lemma 24 establishes Theorem 9.

Discussion. In this paper we consider the HRS problem with a focus on the occupancy-stable matchings. We prove that computing max-sized occupancy-stable matching is NP-hard. Our algorithm gives an approximation guarantee of strictly better than 3 for the max-size occupancy-stable matching. A natural direction is to improve the approximation guarantee and establish the hardness of approximation.

References

1 Atila Abdulkadiroğlu and Tayfun Sönmez. School Choice: A Mechanism Design Approach. *American Economic Review*, 93(3):729–747, 2003. doi:10.1257/000282803322157061.

2 Surender Baswana, Partha Chakrabarti, Sharat Chandran, Yash Kanoria, and Utkarsh Patange. Centralized admissions for engineering colleges in india. *INFORMS Journal on Applied Analytics*, 49(5):338–354, 2019. doi:10.1287/inte.2019.1007.

3 Brian C. Dean, Michel X. Goemans, and Nicole Immorlica. The unsplittable stable marriage problem. In *Fourth IFIP International Conference on Theoretical Computer Science- TCS 2006*, pages 65–75, 2006. doi:10.1007/978-0-387-34735-6_9.

4 David Delacrétaz. Stability in matching markets with sizes. Accessed: 2025-05-05, 2017. URL: https://daviddelacretaz.net/wp-content/uploads/2019/01/DDelacretaz-SMMS-Jan19.pdf.

5 David Delacrétaz, Scott Duke Kominers, and Alexander Teytelboym. Matching mechanisms for refugee resettlement. *American Economic Review*, 113(10):2689–2717, October 2023. doi:10.1257/aer.20210096.

6 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. URL: http://www.jstor.org/stable/2312726.

7 Robert W. Irving, David F. Manlove, and Sandy Scott. The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156(15):2959–2977, 2008. doi:10.1016/j.dam.2008.01.002.

8 Joint Seat Allocation Authority (JoSAA). Josaa - joint seat allocation authority official website. https://josaa.nic.in/, 2025. Accessed: 2025-05-23.

9 Eric J. McDermid and David F. Manlove. Keeping partners together: algorithmic results for the hospitals/residents problem with couples. *Journal of Combinatorial Optimization*, 19(3):279–303, 2010. doi:10.1007/s10878-009-9257-2.

10 Kitty Meeks and Baharak Rastegari. Solving hard stable matching problems involving groups of similar agents. *Theoretical Computer Science*, 844:171–194, 2020. doi:10.1016/j.tcs.2020.08.017.

11 National Resident Matching Program. National resident matching program website. https://www.nrmp.org, 2023. Accessed: 2025-05-23.

12 Nitsan Perach, Julia Polak, and Uriel G Rothblum. A stable matching model with an entrance criterion applied to the assignment of students to dormitories at the technion. *International Journal of Game Theory*, 36:519–535, 2008. doi:10.1007/s00182-007-0083-4.

13 Alvin E. Roth. On the Allocation of Residents to Rural Hospitals: A General Property of Two-Sided Matching Markets. *Econometrica*, 54(2):425–427, 1986. URL: http://www.jstor.org/stable/1913160.

A Omitted proofs

Proof of Lemma 23. Since N is stable, from Lemma 21 the agent a_j^1 should be matched to one of $\{h_j^1, h_a\}$. We divide the proof based on the matched partner of a_j^1 in N.

- Assume $(a_j^1, h_a) \in N$. Here we establish that $T_j^a \subseteq N$. In this case for N to be stable, the hospital h_j^1 has to be fully subscribed in N with agents higher preferred than a_j^1. This implies that $(a_j^5, h_j^1) \in N$. In this case the agent a_j^3 cannot be left unmatched otherwise

(a_j^3, h_j^1) blocks N. Thus, $(a_j^3, h_j^2) \in N$. Since $(a_j^3, h_j^2) \in N$, $(a_j^6, h_j^2) \notin N$. The hospital h_j^1 is matched to a_j^5 and therefore cannot accommodate a_j^4. The matching N cannot leave a_j^4 unmatched, else (a_j^4, h_j^2) blocks N. This implies that $(a_j^4, h_j^2) \in N$. Given the current set of edges in N, if N does not match a_j^2 to h_j^2, then (a_j^2, h_j^2) blocks N. Thus $(a_j^2, h_j^2) \in N$. Thus we have proved that if $(a_j^1, h_a) \in N$ then $T_j^a \in N$.

- Assume $(a_j^1, h_j^1) \in N$. Here, we establish that $T_j^b \subseteq N$. Since a_j^1 occupies unit capacity in h_j^1, a_j^5 cannot occupy h_j^1. Hence, $(a_j^5, h_j^1) \notin N$. Furthermore, $(a_j^3, h_j^1) \in N$ else (a_j^5, h_j^1) blocks N. If a_j^4 is not matched to h_j^1, (a_j^4, h_j^1) blocks N. Therefore, $(a_j^4, h_j^1) \in N$. Given the current set of edges in N, if a_j^6 is not matched to h_j^2 then (a_j^6, h_j^2) block N. So, $(a_j^6, h_j^2) \in N$. From Lemma 21 a_j^2 cannot be unmatched and h_j^2 cannot accommodate a_j^2. Therefore, $(a_j^2, h_b) \in N$. This concludes that $T_j^b \subseteq N$.

This completes the proof. ◀

Proof of Lemma 24. Consider a complete stable matching M of I. Let G be the reduced HRS instance of I. Construct a matching M' as follows, starting with $M' = \phi$.

- For a man m_j with ties in his preference list (w_a, w_b) where $a < b$.
$$M' = M' \cup \begin{cases} T_j^a & \text{if } (m_j, w_a) \in M \\ T_j^b & \text{if } (m_j, w_b) \in M \end{cases}$$
Furthermore for the man m_j, we add $M' = M' \cup S_j^1 \cup S_j^2$.
- For a man m_s with strict preference list in I
$$M' = M' \cup \{(a_s, h_i)\} \cup S_s$$

It can be verified that M' is feasible in G. From Lemma 23, M' is stable.

To prove the other direction, consider a stable matching M' of G. Construct a matching M in I as follows.

- For any j such that m_j is a man with ties in his preference list, (w_a, w_b)
$$M = M \cup \begin{cases} \{(m_j, h_a)\} & \text{if } T_j^a \subseteq M' \\ \{(m_j, h_b)\} & \text{if } T_j^b \subseteq M' \end{cases}$$
- For any s such that m_s is a man with strict preference list.
$$M = M \cup \{(m_s, h_i)\} \quad \text{if } (a_s, h_i) \in M'$$

Since the capacity of h_i is 1, the corresponding woman w_i in I is also matched to at most one man. Every a_j corresponding to agent m_j with ties in his preference list will either be matched to h_a or h_b, so is m_j to m_a or m_b in I. By Lemma 22, a_s has to be matched to one of hospital in $\{h_e, h_f, h_g\}$ in M. So, all men with a strict preference list in I are matched to some woman in I. Therefore, M is feasible and complete in I. Since all men with ties in their preference list are matched, they do not block M. Say, there exists a man-woman pair (m_s, w_i) which blocks M where m_s is a man with a strict preference list. Then corresponding (a_s, h_i) blocks M' in G. Therefore, M is stable in I. ◀

Explorability in Pushdown Automata

Ayaan Bedi
Chennai Mathematical Institute, India

Karoliina Lehtinen
CNRS Researcher, Aix-Marseille Université, LIS, France

Abstract

We study *explorability*, a measure of nondeterminism in pushdown automata, which generalises history-determinism. An automaton is k-explorable if, while reading the input, it suffices to follow k concurrent runs, built step-by-step based only on the input seen so far, to construct an accepting one, if it exists. We show that the class of explorable PDAs lies strictly between history-deterministic and fully nondeterministic PDAs in terms of both expressiveness and succinctness. In fact increasing explorability induces an infinite hierarchy: each level k defines a strictly more expressive class than level $k-1$, yet the entire class remains less expressive than general nondeterministic PDAs. We then introduce a parameterized notion of explorability, where the number of runs may depend on input length, and show that exponential explorability precisely captures the context-free languages. Finally, we prove that explorable PDAs can be doubly exponentially more succinct than history-deterministic ones, and that the succinctness gap between deterministic and 2-explorable PDAs is not recursively enumerable. These results position explorability as a robust and operationally meaningful measure of nondeterminism for pushdown systems.

2012 ACM Subject Classification Theory of computation → Automata extensions

Keywords and phrases Pushdown automata, nondeterminism, explorability, history-determinism

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.12

Funding This work is in part funded by ANR Project QuaSy ANR-23-CE48-0008 and part supported by ReLaX, Research Lab in Computer Science, CNRS IRL 2000.

Acknowledgements We are grateful to Aditya Prakash for his insightful input and helpful discussions.

1 Introduction

Determinism and nondeterminism are central themes in theoretical computer science, particularly in the study of computational models in automata theory and complexity theory. Nondeterminism often brings either increased expressive power or more succinct representations. For example, in the case of finite automata, while nondeterminism does not increase expressive power, it allows for exponentially more succinct representations. In contrast, for Turing machines, while expressiveness remains unchanged, the distinction between deterministic and nondeterministic variants leads to the foundational open problem of the field: *P vs. NP*. Pushdown automata (PDAs), which are strictly more powerful than finite automata, exhibit both increased expressiveness and non recursively ennumerable succinctness when nondeterminism is introduced.

While deterministic models are often seen as overly restrictive, unrestricted nondeterminism functions as a computational "superpower", capable of always making the correct choice. This raises the question of whether is there a meaningful spectrum between determinism and full nondeterminism and how to usefully quantify the degree of nondeterminism.

A variety of measures of nondeterminism have been suggested for pushdown automata. Goldstine et al. quantified nondeterminism by counting the minimum number of guesses (in bits) needed to accept a word, which induces an infinite hierarchy of complexity classes and language subclasses [2], while Han et al. measure computation tree width and ambiguity [4].

© Ayaan Bedi and Karoliina Lehtinen;
licensed under Creative Commons License CC-BY 4.0

Another approach is to restrict nondeterminism to bounded or context-dependent forms [1]. For example, Herzog characterizes pushdown automata with at most k nondeterministic choices as unions of k deterministic context-free languages, noting tradeoffs in descriptive complexity that are sometimes not recursively bounded [10]. Models with regulated nondeterminism (using control sets or stack-content constraints) describe language classes that lie strictly between the deterministic and fully nondeterministic cases.

One well-studied intermediate form of nondeterminism is *bounded ambiguity*, which restricts the number of accepting runs per input – for instance, to at most one (unambiguity), or at most k, for some fixed constant k. In parallel, another notable form of restricted nondeterminism that simplifies decision problems like universality and game solving is *history-determinism* (HD), also known as *good-for-games* (GFG) nondeterminism [9]. The key idea behind HD automata is that their nondeterministic choices can be resolved in a forward, step-by-step manner, based only on the input processed so far, without any need to anticipate future input symbols [3].

Recently, history-deterministic pushdown automata (HD-PDAs) have been shown to be more expressive than their deterministic counterparts [3]. Moreover, HD-PDAs were shown to enjoy at least exponential succinctness over deterministic PDAs. However, when compared to general nondeterministic PDAs, HD-PDAs are strictly less expressive and also less succinct, by at least doubly exponential factor [3]. Furthermore, at least one of these succinctness gaps is far from tight, since the succinctness gap between nondeterministic and deterministic PDAs is not recursively enumerable (RE) [6] – the question of which one, or both, was left open.

This significant gap between HD-PDAs and general nondeterministic PDAs motivates the exploration of intermediate classes of languages. By identifying and characterizing such classes, we aim to better understand the trade-offs between expressiveness, succinctness, and the degree of nondeterminism permitted in pushdown computations. Therefore we study the notion of *explorability*, which was introduced for finite state automata by Hazard and Kuperberg [8] as a generalisation of history-determinism. Intuitively, k-explorability corresponds to having k separate runs that are being resolved on-the-fly, with the requirement that for any word in the language, at least one of the runs should be accepting.

The problem of recognizing explorable NFAs is ExpTime-complete for automata over finite words and for Büchi automata. In the setting of infinite words, recognizing ω-explorable automata is also ExpTime-complete for safety and co-Büchi acceptance conditions [8]. Moreover Hazard, Idir and Kuperberg showed that there is a trade-off between the explorability of an automaton and the number of priorities needed to recognise all ω-regular languages [7].

Our contributions. In this work, we extend the study of PDAs by investigating their expressiveness and succinctness across increasing levels of explorability. We show that *explorable* PDAs are strictly more expressive than HD-PDAs. Furthermore, we establish an infinite hierarchy within the class of explorable PDAs: k-explorable PDAs are strictly less expressive than $(k+1)$-explorable ones. We also prove that the class of explorable PDAs is strictly less expressive than the class of general nondeterministic PDAs.

We then ask under what conditions does the class of explorable PDAs match the expressive power of the class of fully nondeterministic PDAs? We answer this by introducing a *parameterized* notion of explorability, where the number of concurrent runs may depend on the length of the input word. In this setting, we demonstrate a hierarchy between constant-bounded, linearly-bounded, and exponentially-bounded explorability. Exponentially explorable PDAs capture exactly the class of context-free languages, and are therefore

equivalent in expressiveness to general nondeterministic PDAs. While we only study this notion over PDAs, the notion of parameterised explorability might be of independent interest, as is the case for parameterised ambiguity.

We then turn our attention to the succinctness of explorable PDAs. We prove that the succinctness gap between HD-PDAs and general explorable PDAs is at least doubly exponential. Moreover, we establish that a similar doubly exponential gap persists even when explorability is restricted to a constant. Delving further into the non-RE separation between deterministic and nondeterministic models, we show that the succinctness gap between DPDAs and 2-explorable PDAs is not recursively enumerable. That is, there exists no recursive bound on the size blow-up required to simulate certain 2-explorable PDAs by DPDAs.

2 Preliminaries

Pushdown Automaton

▶ **Definition 1.** *A* pushdown automaton *(PDA) is a computational model that extends finite automata with a stack, providing additional memory. Formally, a PDA is a tuple* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ *where:*

- Q *is a finite set of states,*
- Σ *a finite input alphabet,*
- Γ *a finite stack alphabet,*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^{\leq 2}}$ *is the transition function,*
- $q_0 \in Q$ *is the initial state,*
- $Z_0 \in \Gamma$ *is the initial stack symbol, and*
- $F \subseteq Q$ *is the set of accepting states.*

At any point during computation, the *mode* of a PDA is of the form $Q \times \Gamma$ and is determined by its current state and the symbol at the top of the stack. The behavior of the PDA is defined by the transition function δ, which may allow multiple transitions from the same mode. The transitions enabled at a mode, runs, accepting runs, and language of an automaton are defined as usual [11]. The *size* of a PDA is the product of the number of states $|Q|$ and the number of stack symbols $|\Gamma|$.

PDAs accept exactly the class of languages known as *context-free languages* (CFLs). A PDA is said to be *deterministic* if for all $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, and $X \in \Gamma$, the set $\delta(q, a, X)$ contains at most one element, and if $\delta(q, \varepsilon, X) \neq \emptyset$, then $\delta(q, a, X) = \emptyset$ for all $a \in \Sigma$. The class of languages accepted by deterministic PDAs (DPDAs) is called the class of *deterministic context-free languages* (DCFLs), and it is known that DCFL $\subsetneq$ CFL, i.e., DCFLs form a strict subset of CFLs [11].

3 Explorability of Pushdown Automata

The *k*-Explorability Game for Pushdown Automata

As described by Hazard et al. [8], *explorability* restricts or measures the amount of nondeterminism an automaton needs to accept a language, while allowing more flexibility than history determinism (HD). For a given $k \in \mathbb{N}$, an automaton is said to be *k-explorable* if, when processing an input, it is sufficient to keep track of most k runs in order to construct an accepting run, if one exists. This generalizes the idea of HD automata, which corresponds to the special case where $k = 1$.

We now introduce the *k-explorability game* [8] over a PDA, which generalises the game-based definition of HD-automata, and captures a hierarchy of nondeterministic behavior in pushdown automata (PDAs). This game is played between two players – *Spoiler* and *Determiner* – on a nondeterministic PDA A.

▶ **Definition 2** (*k*-Explorability Game for PDAs). *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a pushdown automaton, and let $k \in \mathbb{N}$ be a fixed integer. The* k-explorability game *on P is played on the arena $Q^k \times (\Gamma^*)^k$, where each of k tokens maintains its own copy of the control state and stack content. We can imagine a configuration here as k tokens say 1... k placed in the space $Q \times \Gamma^*$ and arena is the set of all possible configurations.*

The game proceeds as follows:

- ***Initialization:*** *The initial configuration is a k-tuple of identical PDA configurations:*

$$S_0 = ((q_0, Z_0), \ldots, (q_0, Z_0))$$

- ***Gameplay:*** *At each step $i \geq 1$, from configuration S_{i-1}:*
  1. ***Spoiler*** *chooses an input symbol $a_i \in \Sigma$.*
  2. ***Determiner*** *responds by selecting the next configuration*

$$S_i = ((q_i^1, \gamma_i^1), \ldots, (q_i^k, \gamma_i^k)) \in (Q \times \Gamma^*)^k$$

such that for each $l \in \{1, \ldots, k\}$, there exists a path of transitions in P from $(q_{i-1}^l, \gamma_{i-1}^l)$ to (q_i^l, γ_i^l) of the form:

$$(q_{i-1}^l, \gamma_{i-1}^l) \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{a_i} (q_i^l, \gamma_i^l)$$

where each step is valid under the PDA transition relation δ.

- *The play continues for an infinite number of steps and a word $w = a_1 a_2 \ldots$ is formed.*

A play *is an infinite sequence $\pi = S_0\, t_1\, S_1\, t_2\, S_2\, \ldots$ such that: $S_0 = ((q_0, Z_0), \ldots, (q_0, Z_0)) \in (Q \times \Gamma^*)^k$, is a k-tuple of initial state q_0 and initial stack configuration Z_0. And for all $i \geq 1$, and for each $l \in \{1, \ldots, k\}$, the transition $(q_{i-1}^l, \gamma_{i-1}^l) \xrightarrow{t_i^l} (q_i^l, \gamma_i^l)$ is valid according to the PDA's transition relation δ.*

The play is won by Determiner *if for every finite prefix of $w = a_1 a_2 \ldots$ that is in the language of L, i.e. $w' = a_1 \ldots a_n \in L(P)$, there exists at least one token $l \in \{1, \ldots, k\}$ such that the sequence of configurations for token l forms an accepting run of P on w'. Otherwise,* Spoiler *wins the game.*

We say that a PDA P is *k-explorable* (Expl_k-PDA) if Determiner has a winning strategy in the k-explorability game on P. If P is k-explorable for some $k \in \mathbb{N}$, we say that P is *explorable* (Expl-PDA). When $k = 1$ then it's exactly the game characterising HD ([9, Section 3]) and therefore HD is 1-explorable.

▶ **Definition 3.** *We define* Expl_k-CFL *as the class of languages recognized by k-explorable nondeterministic pushdown automata. That is,*

$$\mathsf{Expl}_k\text{-CFL} = \{L \subseteq \Sigma^* \mid L \text{ is recognized by a } k\text{-explorable PDA}\}.$$

▶ **Definition 4.** *We define* Expl-CFL *as the class of languages recognized by* explorable *pushdown automata. Formally,* $\mathsf{Expl}\text{-CFL} = \bigcup_{k \in \mathbb{N}} \mathsf{Expl}_k\text{-CFL}$.

▶ **Definition 5** (*k*-Run in a *k*-Explorable PDA). *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a nondeterministic pushdown automaton, and let $k \in \mathbb{N}$. A* k-run *of P on an input word $w = a_1 a_2 \dots a_n \in \Sigma^*$ in the k-explorability setting is a sequence of k-tuples of configurations:*

$$\mathcal{R} = (C_0^1, \dots, C_0^k), (C_1^1, \dots, C_1^k), \dots, (C_n^1, \dots, C_n^k)$$

such that:

- *For all $1 \leq l \leq k$, $C_0^l = (q_0, Z_0)$ is the initial configuration (same across all tokens),*
- *For each step $i = 1, \dots, n$ and each token $l = 1, \dots, k$, there exists a valid PDA path:*

$$C_{i-1}^l \xrightarrow{a_i} C_i^l,$$

A k-run $\mathcal{R}$ is said to be accepting *if there exists at least one token $l \in \{1, \dots, k\}$ such that the final configuration $C_n^l = (q_n^l, \gamma_n^l)$ satisfies $q_n^l \in F$ (i.e., the control state is accepting).*

Parameterised Explorability

The notion of k-explorability captures bounded nondeterminism by allowing the Determiner to explore at most k computational branches in parallel. However, in many settings, the amount of explorability required to resolve nondeterminism may grow with the size of the input. To capture this, we extend the explorability to a *parameterized* framework, where the number of tokens available to Determiner is not fixed but instead given by a function $f(n)$ of the input length n.

▶ **Definition 6** ($f(n)$-Parameterized Explorability Game). *Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a nondeterministic pushdown automaton, and let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. The* $f(n)$-parameterized explorability game *on P is played between two players,* ***Spoiler*** *and* ***Determiner****, as follows:*

- ***Initialization:*** *Spoiler announces a number $n \in \mathbb{N}$, declaring the maximum length of the word he will play. Determiner is then given $f(n)$ tokens, each representing a copy of the PDA configuration.*
- ***Gameplay:*** *is similar to as defined for k-Explorability Game.*

The play is won by Determiner *if for every prefix of the Spoiler's word in $w \in \Sigma^{\leq n} \cap L(P)$, there exists at least one token $l \in \{1, \dots, f(n)\}$ such that the sequence of configurations for token l forms an accepting run of P on w. Otherwise, Spoiler wins.*

A PDA P is said to be *$f(n)$-explorable* if Determiner has a winning strategy in the $f(n)$-parameterized explorability game on P. Let C be a class of functions (such as $O(n)$); we define the language class:

$$\mathsf{Expl}_C\text{-}\mathsf{CFL} = \{L \subseteq \Sigma^* \mid \exists \text{ an } f(n)\text{-explorable PDA } P \text{ such that } L = L(P) \text{ and } f(n) \in C\}.$$

4 Expressiveness

It is well known that deterministic pushdown automata (DPDA) recognize the class of deterministic context-free languages (DCFL), which is strictly contained within the class of context-free languages (CFL) recognized by nondeterministic pushdown automata (PDA). Recent work by Guha et al. [3] shows that HD-PDA defines a language class (HD-CFL) that properly extends DCFL but is still strictly contained within CFL.

▶ **Theorem 7** (Theorem 4.1 [3]). *DCFL $\subsetneq$ HD-CFL $\subsetneq$ CFL*

We now turn our attention to the expressive power of *explorable pushdown automata*. Analogous to the results on history-deterministic pushdown automata, we investigate the hierarchy induced by allowing increasing, but bounded, explorability. In particular, we show that k-explorability induces an infinite hierarchy, which is strictly contained within CFL:

▶ **Theorem 8.** *For all k,* $\mathsf{Expl}_k\text{-CFL} \subsetneq \mathsf{Expl}_{k+1}\text{-CFL} \subsetneq \mathit{CFL}$.

4.1 Expl_k-CFL $\subsetneq$ Expl_{k+1}-CFL

To prove that $(k+1)$-explorable pushdown automata are strictly more expressive than k-explorable ones, we consider, for each $i \in \{1, \ldots, k\}$, the language $L_i := \{a^n b^{in} \mid n \in \mathbb{N}\}$. We then define the language $L := \bigcup_{i=1}^{k+1} L_i$.

We claim that L is accepted by a $k+1$-explorable pushdown automaton but cannot be recognized by any k-explorable pushdown automaton. Intuitively, accepting L requires distinguishing among k different linear relations between the number of a's and b's, each corresponding to a different multiplicative factor i. This requires k distinct tokens, one for each L_i, which cannot be simulated with fewer runs.

▶ **Lemma 9.** *The language L is recognized by a $(k+1)$-explorable pushdown automaton.*

Proof. For each i, L_i is recognized by a DPDA D_i that pushes i symbols onto the stack at each a and then compares the number of b's to the stack height by popping the stack as it reads b. The automaton that initially nondeterministically chooses among ε-transitions to the initial states of each D_i for $i \in [1..k+1]$ clearly recognises L and is $k+1$-explorable since Determiner can win the $k+1$-explorability game by having each D_i explored by a distinct token. ◀

L is not k-explorable

We observe that a k-explorable PDA P would have to accept some words $a^n b^\ell$ and their continuation $a^n b^{\ell'}$ along the same run, as Determiner does not have enough tokens to check membership to each L_i with a disjoint run. Therefore, there would be some accepting runs that can be extended into another accepting run via more b-transitions. Then, we build a PDA P' which is similar to P, except that after seeing an accepting state, it can also read c's instead of b's. We show, using a pumping argument, that this automaton would recognise a language that is not CFL, leading to a contradiction. The difficulty in this proof is that we do not have an exact description for the language recognised by P' as it depends on exactly which pairs of words P accepts along a single run, and the hypothesis of k-explorability only tells us that this must occur for some pairs ℓ, ℓ' for each n. However, we can describe $L(P')$ in enough detail to show that it is, in any case, not CFL.

We define the language $L_{i,j,n} := \{a^n b^{in} c^{jn}\}$, where $i, j, n \in \mathbb{N}$. Each $L_{i,j,n}$ consists of strings with block sizes linearly dependent on n, with coefficients i and j for the lengths of the b and c blocks, respectively.

Now we define a language L_S such that for each $n \in \mathbb{N}$, the set of words of the form $a^n b^m c^k \in L'$ is contained in a finite and nonempty union of such languages $L_{i,j}, n$, but the choice of the coefficents is and js may vary with n. Formally, given $S = (S_n)_{n \in \mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty.

$$L_S := \bigcup_{n \in \mathbb{N}} \left(\{a^n\} \cdot \bigcup_{(i,j) \in S_n} \{b^{in} c^{jn}\} \right),$$

That is, for every $n \in \mathbb{N}$, L_S consists of a finite union of languages of the form $L_{i,j,n}$. In this way, L_S represents a "non-uniform" or "n-dependent" union of linearly structured languages.

▶ **Lemma 10.** *Given $S = (S_n)_{n\in\mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty, the language L_S is not context-free.*

The proof is a pumping argument, detailed in the Appendix.

▶ **Lemma 11.** *The language $L := \bigcup_{i=1}^{k+1} L_i$, where $L_i := \{a^n b^{in} \mid n \in \mathbb{N}\}$, is not recognized by any k-explorable PDA.*

Proof. Assume, towards contradiction, that there exists a k-explorable PDA $\mathcal{A}$ that recognizes L. Since L includes strings of the form $a^n b^{in}$ for $i \in \{1, 2, \ldots, k+1\}$, and we only have k tokens, by the pigeonhole principle, for any fixed n, for any winning Determiner strategy, there must be at least one token that accepts two different strings $a^n b^{in}$ and $a^n b^{jn}$ for some $i < j$. That means that the run built by this token visits an accepting state at least twice. We define S_n as the pairs i, j such that there is an accepting run over $a^n b^{jn}$ of which a prefix accepts $a^n b^{in}$. By the pigeonhole principle, that S_n is non-empty for all n.

We use this fact to construct a new PDA $\mathcal{A}_c$ that accepts the language L_S, for $S = (S_n)_{n\in\mathbb{N}}$ as described above. As previously shown, L_S is not context-free, so the existence of such a PDA will lead to a contradiction.

Let $\mathcal{A} = (Q, \Sigma, \Gamma, q_I, \Delta, F)$, with $Q = \{q_0, q_1, \ldots, q_n\}$ and $q_I = q_0$. We define a modified PDA $\mathcal{A}_c = (Q \cup Q', \Sigma', \Gamma, q_I, \Delta', F')$, where:

- $\Sigma' = \{a, b, c\}$,
- $Q' = \{q'_0, q'_1, \ldots, q'_n\}$ is a copy of Q,
- $F' = \{q'_f \mid q_f \in F\}$,
- $\Delta' = \Delta \cup \Delta_c$, with Δ_c defined as:
  1. For every $q_f \in F$, add $(q_f, X, \varepsilon, q'_f, X)$ for all $X \in \Gamma_\perp$ (transferring control to the new state space),
  2. For every transition $(q_i, X, b, q_j, \gamma) \in \Delta$, add $(q'_i, X, c, q'_j, \gamma) \in \Delta_c$ – replacing the input symbol b with c in the copied state space.
  3. For every $(q_i, X, \varepsilon, q_j, \gamma) \in \Delta$, add $(q'_i, X, \varepsilon, q'_j, \gamma) \in \Delta_c$. - to replicated ε-transitions

Intuitively, the modified PDA simulates the original input $a^n b^{in}$ on $\mathcal{A}$, and then, upon acceptance, continues in the copied state space simulating a second phase where each b transition is replaced by a c-transition, thereby accepting strings of the form $a^n b^{in} c^{(j-i)n}$.

For each pair in some S_n, there is an accepting run in P that extends into another accepting run. Therefore P' accepts the word $a^n b^{in} c^{(j-i)n}$ for each $(i, j) \in S_n$. Hence A_c accepts $L_S \cup L$. This implies that L_S is context-free (by intersection with the regular language $a^* b^* c^*$) – a contradiction, since we have already shown in Lemma 4.2 that $L_S \notin \mathsf{CFL}$.

Thus, no k-explorable PDA can recognize L. ◀

We conclude the first half of **Theorem 8**: For all $k \in \mathbb{N}$, the class of languages recognized by $(k+1)$-explorable pushdown automata strictly contains the class of languages recognized by k-explorable pushdown automata. That is, $\mathsf{Expl}_k\text{-}\mathsf{CFL} \subsetneq \mathsf{Expl}_{k+1}\text{-}\mathsf{CFL}$.

4.2 Expl-CFL $\subsetneq$ CFL

To separate the class of explorable context-free languages from the full class of context-free languages, we consider the following language:

$$L_{block} := \{(a^*\#)^* b^n \mid n = \text{length of some } a\text{-block}\}.$$

Intuitively, L_{block} consists of strings formed by a sequence of a-blocks separated by $\#$, followed by a block of b's whose length matches the length of some previous a-block. For example, the string $a^3\#a^5\#a^2\#b^5 \in L_{block}$, since the final b-block has length 5, which matches the length of the second a-block.

While L_{block} is clearly CFL, the intuition is that each of Determiner's tokens can only compare one block of a's to the final block of b's, and as the number of a-blocks is unbounded, so is the number of tokens that she would need to win the explorability game, making this language not explorable.

▶ **Lemma 12.** $L_{block} \in \boldsymbol{CFL}$

Proof. A nondeterministic PDA can guess and store the length of one of the a-blocks on the stack, then skip the rest of the input until it reaches the b-block, and verify the count by popping the stack. ◀

▶ **Lemma 13.** *The language* $L_{block} := \{(a^*\#)^*b^n \mid n = \text{length of some } a\text{-block}\}$ *is not recognized by an explorable PDA.*

Proof. Assume, towards a contradiction, that for some k, $L_{block} \in \mathsf{Expl}_k\text{-}\mathsf{CFL}$, i.e., it is recognized by an k-explorable PDA. Observe that $\mathsf{Expl}_k\text{-}\mathsf{CFL}$ is closed under intersection with regular languages since taking a product with a deterministic finite automaton does not increase the explorability of the PDA.

Let us take the regular language $R = (a\#)^{k+1}b^*$. Then the intersection

$$L_{k-block} := L_{block} \cap R = \left\{(a^*\#)^{k+1}b^n \mid n = \text{length of some } a\text{-block}\right\}$$

must also be in $\mathsf{Expl}_k\text{-}\mathsf{CFL}$ under our assumption.

So the problem comes down to showing that $L_{k-block} \notin \mathsf{Expl}_k\text{-}\mathsf{CFL}$. Assume, towards a contradiction, that $L_{k-block}$ is recognized by a k-explorable PDA A. Let w be a word in $(a^*\#)^{k+1}$. In the k-explorability game, since there are $k+1$ a-blocks but only k tokens, if all the blocks are of distinct lengths then, at least one token will accept both a word and its prefix. Therefore, for each such w, there are some accepting runs that can be extended into another accepting run via more b-transitions. Let S'_w be the set of pairs of integers such that both wb^i and wb^j are accepted along the same run. Note that for each $(i,j) \in S'_w$, both i and j match the length of some a-block in w and $j > i$. Call $S' = (S'_w)_{w\in W}$ where W consists of $(a^*\#)^{k+1}$ such that S'_w is non-empty. Note that S'_w is necessarily non-empty for w in which the block lengths are all distinct, and may or may not be empty otherwise.

As in Lemma 11 we can then build a second PDA, in which ε-transitions move from the final states of a first copy of A to the same state in another copy, in which b-transitions are replaced by c-transitions. Taking the product with a DFA for $(a^*\#)^{k+1}b * c*$, this results in a PDA A' for the language:

$$L_{block-S'} := \left\{wb^ic^{i-j} \mid w \in (a^*\#)^{k+1}, (i,j) \in S'_w\right\}.$$

By construction, if $L_{k-block} \in \mathsf{Expl}_k\text{-}\mathsf{CFL}$, then $L_{block-S'}$ is accepted by the A'. However, from Lemma 14, this language is not CFL. This contradicts that it is recognized by a PDA.

Hence, $L_{k-block} \notin \mathsf{Expl}_k\text{-}\mathsf{CFL}$, and therefore $L_{block} \notin \mathsf{Expl}_k\text{-}\mathsf{CFL}$ as well. ◀

▶ **Lemma 14.** *Given* $S' = (S'_w)_{w\in(a^*\#)^{k+1}}$ *in which some* S'_w *are non-empty, then the language* $L_{block-S'}$ *is not context-free.*

The proof, detailed in the appendix, consists of a pumping argument.

▶ **Theorem 15.** *Expl-CFL* $\subsetneq$ *CFL*

Proof. Follows directly from Lemma 12 and Lemma 13: we have exhibited a context-free language L_{block} which is not in $\mathsf{Expl}_k\text{-}\mathsf{CFL}$, for any k, establishing the strict inclusion. ◀

5 Parametrized Explorability and Its Expressiveness

In the previous section, we established the existence of an expressiveness gap across different fixed levels of exploration. However, in that analysis, the exploration was treated as a constant, independent of the input. In this section, we investigate the consequences of parameterising the level of exploration as a function of the input length, as characterised by the parameterised explorability game. This allows us to study how expressiveness scales when the exploration level is allowed to grow with the size of the input.

We are led to the following conjecture, which captures the hierarchy induced by varying exploration bounds:

$$\mathsf{Expl}_{O(1)}\text{-}\mathsf{CFL} \subsetneq \mathsf{Expl}_{O(n)}\text{-}\mathsf{CFL} \subsetneq \mathsf{Expl}_{exp}\text{-}\mathsf{CFL} = \mathsf{CFL}$$

Here, we write *exp* for the set of exponential functions, that is, in $O(2^{p(n)})$ for some polynomial p. Intuitively $\mathsf{Expl}_{O(f(n))}\text{-}\mathsf{CFL}$ is the class of languages recognised by a pushdown automaton that is allowed to have at most $O(f(n))$ tokens for *exploration*, where n is the length of the input. Observe that $\mathsf{Expl}_{O(1)}\text{-}\mathsf{CFL}$ is precisely the the class Expl-CFL discussed so far.

The conjecture above posits a strict hierarchy of language classes recognized by explorable pushdown automata under increasing exploration bounds. At the lower end, constant-bounded exploration ($O(1)$) is provably limited in expressiveness, while allowing exploration to grow linearly with input length ($O(n)$) yields strictly greater power. Ultimately, permitting exponential exploration suffices to recover the full class of context-free languages.

▶ **Lemma 16.** $\mathsf{Expl}_{O(1)}\text{-}\mathsf{CFL} \subsetneq \mathsf{Expl}_{O(n)}\text{-}\mathsf{CFL}$

Proof. In the previous section, we saw that the language

$$L_{block} := \{(a^*\#)^* b^n \mid n = \text{length of some } a\text{-block}\}$$

cannot be recognized by any k-explorable pushdown automaton for any $k \in \mathbb{N}$ (Lemma 13). Consequently, we conclude that $L_{block} \notin \mathsf{Expl}_{O(1)}\text{-}\mathsf{CFL}$.

However, it is straightforward to observe that $L_{block} \in \mathsf{Expl}_{O(n)}\text{-}\mathsf{CFL}$. Indeed, we can construst a PDA that nondeterministically chooses an a-block and compares it with the b-block. Since the total number of such a-blocks is bounded by the length of the input word, it suffices to use $O(n)$ runs to cover all possible cases. Within each run, the automaton can compare the length of an a-block with the number of b's at the end of the input. If a match is found, the input is accepted. Hence, the inclusion $\mathsf{Expl}_{O(1)}\text{-}\mathsf{CFL} \subsetneq \mathsf{Expl}_{O(n)}\text{-}\mathsf{CFL}$ is strict. ◀

▶ **Lemma 17.** $\mathsf{Expl}_{\mathrm{exp}}\text{-}\mathsf{CFL} = \mathsf{CFL}$

Proof. Let $L \in \mathsf{CFL}$. Then there exists a PDA M that accepts L. It is well known that for any PDA, there exists an equivalent PDA that accepts the same language but does not use ε-transitions[13], so w.l.o.g. we assume M to have no ε-transitions.

Let m be the maximum number of nondeterministic transitions possible from any state in M'. Consider an input word w of length n. Since the machine can make at most m choices at each step, the number of possible runs of M' on input w is at most m^n.

Therefore, the total number of possible runs grows at most exponentially with the length of the input. Therefore exponentially many tokens are enough to explore each of the possible runs. Hence, $\mathsf{CFL} \subseteq \mathsf{Expl}_{\mathrm{exp}}\text{-}\mathsf{CFL}$.

The reverse inclusion is trivial. Therefore, $\mathsf{CFL} = \mathsf{Expl}_{\mathrm{exp}}\text{-}\mathsf{CFL}$. ◀

6 Succinctness

We now shift our focus to the succinctness of pushdown automata. Recall that the size of a PDA is the product $|Q| \cdot |\Gamma|$, where Q is the set of states and Γ is the stack alphabet. We argue that significant succinctness gaps exist between different levels of exploration. We build on several results from the work of Guha et al. [3].

We demonstrate that *explorable* PDAs are not only more expressive than history-deterministic PDAs (HD-PDAs) and deterministic PDAs (DPDAs), but also more succinct. Furthermore, we show that fixing the degree of exploration yields a *double-exponential gap* in succinctness when compared to unrestricted explorable PDAs.

6.1 Explorable vs. HD-PDA

Guha et al [3] showed that a nondeterministic pushdown automaton (PDA) can recognize the language $L_n = (0+1)^*1(0+1)^{n-1}$ using only $O(\log(n))$ space, while any history-deterministic PDA (HD-PDA) requires space exponential in n. This establishes a double-exponential gap in succinctness between PDAs and HD-PDAs.

In this work, we demonstrate that the same double exponential gap occurs already between HD-PDAs and explorable PDAs. Specifically, we show that an explorable PDA can recognize L_n using only $O(\log(n))$ space, highlighting the succinctness advantage of explorability over history-determinism.

▶ **Lemma 18.** *There exists an explorable PDA of size $O(\log n)$ recognising Ln.*

Proof. Guha et al. [3] described a PDA P_n that recognises L_n in $O(\log(n))$ space. We adjust their construction to get $\mathcal{P}'_n$ that is also n-explorable. The PDA $\mathcal{P}_n$ nondeterministically guesses the nth bit from the end of the input, checks that it is a 1 or a 0 and stored that in the state space, then switches to a counting gadget that verifies the input ends in exactly n steps, as follows:

1. It pushes the binary representation of $n-2$ onto the stack. For example, if $n = 8$, then 110 is pushed onto the stack with 0 at the top. Note that $\log(n-2)$ states suffice for pushing the binary representation of $n-2$. If $n = 1$, then instead of pushing anything onto the stack, the automaton directly moves to a final state without any enabled transitions.
2. Then $\mathcal{P}_n$ moves to a state that attempts to decrement the counter by one for each successive input letter, as follows: when an input letter is processed, it pops 0s until a 1 is at the top of the stack, say m 0s. Then, it replaces the 1 with a 0, and finally pushes m 1s back onto the stack before processing the next letter. If the stack empties before a 1 is found, then the counter value is 0, and the automaton moves to an accepting state or dummy state depending on the initial 1-check. Note that $O(\log n)$ states again suffice for this step.
3. To make the automaton P'_n, we keep the description of P_n and add an ε-transition from the final state and dummy state (as deribed above) back to the initial state.

Now, consider the n tokens, numbered $0, 1, \ldots, n-1$, and an input word $w = w_1 w_2 \ldots w_m$. Each token (i) starts the logarithmic counting at input positions j such that $j \bmod n = i$. Therefore, if the input contains a 1 at the nth position from the end, then there exists at

least one token that reaches the final state of the PDA. Furthermore, if an additional input symbol is read after reaching the final state, the ε-transition takes the token to the initial state, and the log-space counter is reinitialized. This allows the token to repeat the counting procedure from the next position.

Thus, $\mathcal{P}'_n$ has $O(\log n)$ states and is n-explorable. Note that for all n, $\mathcal{P}'_n$ uses only three stack symbols: 0, 1, and $\perp$. Altogether, the size of $\mathcal{P}'_n$ is $O(\log n)$, and $\mathcal{P}'_n$ recognises L_n. ◀

We can now state the following succinctness result.

▶ **Theorem 19.** Expl-CFL *can be double-exponentially more succinct than* HD-PDA.

6.2 k-Explorable PDA vs. Explorable PDA

We now compare the succinctness gap between PDA of bounded and unbounded explorability, using the following family of languages.

$$L_{mod\ n} := (0+1)^{<n}(1(0+1)^{n-1})^*$$

That is, a word belongs to $L_{mod\ n}$ if every position that is a multiple of n, counting from the end, contains a 1.

▶ **Lemma 20.** *Any k-explorable pushdown automaton (PDA) that accepts the language $L_{mod\ n}$ must have size at least $\frac{\sqrt[k]{2^n}}{n}$*

Proof. We define the *stack height* of a configuration $c = (q, \gamma)$ as $\mathrm{sh}(c) = |\gamma| - 1$, and we define *steps* of a run as follows. Consider a run $c_0\tau_0c_1\tau_1 \cdots c_{n-1}\tau_{n-1}c_n$. A position $s \in \{0, \ldots, n\}$ is a *step* if for all $s' \geq s$, we have that $\mathrm{sh}(c_{s'}) \geq \mathrm{sh}(c_s)$, that is, the stack height is always at least $\mathrm{sh}(c_s)$ after position s. Any infinite run of a PDA has infinitely many steps. We have the following observation:

▶ Remark 21. If two runs of a PDA have steps s_0 and s_1, respectively, with the same mode, then the suffix of the run following the step s_0 can replace the suffix of the other run following the step s_1, and the resulting run is a valid run of the PDA.

Consider an infinite run of a pushdown automaton (PDA) P on some input word w^ω where w is of length n. Each position of the run is characterised by an *indexed mode*, which is a tuple consisting of the current state, the top symbol of the stack, and the current position in w (an integer in $[0, n-1]$). Since the set of states Q, the stack alphabet Γ, and the input positions $[0, n]$ are all finite, the total number of distinct indexed modes is bounded by $|Q| \cdot |\Gamma| \cdot n$.

Since an infinite run must contain infinitely many steps and there are only finitely many indexed modes, by the pigeonhole principle, some indexed mode must appear infinitely often as a step. That is, there exists a mode $m = (q, \gamma, i)$, where $q \in Q$, $\gamma \in \Gamma$, and $i \in \{0, 1, \ldots, n\}$, such that m occurs at infinitely many points in the run as a step.

Now consider a k-explorable pushdown automaton. Determiner's strategy witnessing this induces a tuple of k runs in parallel on a given input word. As in the previous argument, in each individual run, some indexed mode must appear infinitely often as a step. Since there are at most $|Q| \cdot |\Gamma| \cdot n$ possible indexed modes for a single run, the number of possible combinations of indexed modes across all k runs is at most $(|Q| \cdot |\Gamma| \cdot n)^k$.

This gives an upper bound on the number of distinct indexed mode combinations that can occur as a step infinitely many times in a k-run of a k-explorable PDA.

Now, observe that there are 2^n distinct binary words of length n. Suppose the number of distinct modes is strictly less than $\frac{\sqrt[k]{2^n}}{n}$. Then, by the pigeonhole principle, there must exist at least two distinct words, say w_1 and w_2, such that the infinite k-run (k-tuple of

infinite runs) $\overline{\rho}_1$ and $\overline{\rho}_2$, induced by the k-explorable PDA on inputs w_1^ω and w_2^ω, respectively, contain infinitely many steps at the same index modulo n, and with identical step modes. Let these step modes be $m_1, m_2, \ldots, m_k$, one for each of the k runs for both w_1 and w_2, occurring at positions $s_1, s_2, \ldots, s_k$ in $\overline{\rho}_1$, and at corresponding positions $s'_1, s'_2, \ldots, s'_k$ in $\overline{\rho}_2$, such that for each $\alpha \in [k]$, we have

$$s_\alpha \bmod n = s'_\alpha \bmod n \quad \& \quad |s_\alpha| > n \quad \& \quad |s'_\alpha| > n$$

Since w_1 and w_2 are distinct words, they differ at some position, say j. WLOG, assume that $w_1[j] = 1$ while $w_2[j] = 0$. For w_1 at every instance where the input length modulo the word length (n) is $j-1$, one of runs in $\overline{\rho}_1$ must reach an accepting state (since the bit at position j in w_1 is 1). However, in the run ρ_2, at every such instance (i.e., when the position modulo n is $j-1$), no accepting state can be reached in $\overline{\rho}_2$, since the corresponding bit in w_2 is 0.

Let $s = max(s_1, \ldots, s_k)$, and consider a position p where the modulo n is $j-1$ and $p > s$. At position p, ρ_1 should have an accepting state in one of the explorable runs as explained before. Now, the contradiction is as follows: WLOG, assume that the accepting state is at the first run in $\overline{\rho}_1$. This means that the suffix of first run $\overline{\rho}_2$ starting from position $s'_1 + 1$ can be replaced with the suffix of first run in $\overline{\rho}_1$ starting from position $s_1 + 1$ and this will yield a valid run (by remark 4.1.1) and accepting run (see figure 1). After replacement, the resultant run is on a word which is some concatenation of substrings of w_2^* and w_1^*; however since s_1 and s'_1 share the same index, the index of the final state also remains the same. As $s'_1 > n$ the initial occurrences of $w_2[j]$ will lie at some (0 *mod* n)th bit from the end; therefore, the resultant word is not in $L_{mod\ n}$. However, after the replacement, P accept the word, leading to a contradiction. ◀

▶ **Theorem 22.** Expl-CFL *can be double-exponentially more succinct than* Expl_k-CFL.

Proof. As in the Lemma 18, an explorable PDA (more specifically an n-explorable PDA) can recognise $L_{mod\ n}$ using $\log(n)$ space. In contrast, we showed that for each fixed k, any k-explorable PDA recognizing $L_{mod\ n}$ must use at least $\frac{\sqrt[k]{2^n}}{n}$ size. This establishes a double exponential gap in succinctness between fully explorable PDAs and their k-explorable counterparts. ◀

6.3 Deterministic vs. Explorable PDA

Theorem 5.1 in the work of Guha et al. [3] establishes that HD-PDAs can be exponentially more succinct than DPDAs. Since HD-PDAs correspond precisely to 1-explorable PDAs, it follows that the same exponential succinctness gap holds between DPDAs and explorable PDAs. Hartmanis [6] proved that the relative succinctness of representing deterministic context-free languages (DCFLs) using deterministic versus nondeterministic pushdown automata (PDAs) is not bounded by any recursive function. In this work, we adapt and extend his proof technique to establish a similar result for deterministic and explorable PDAs – showing that the succinctness gap between them is likewise non-recursive.

Let M be a Turing machine, for an input x, let $\mathrm{ID}_0(x)$ denote the initial configuration of M on input x, and let $\mathrm{ID}_1(x), \mathrm{ID}_2(x), \ldots$ denote the successive configurations during the computation.

If $x = a_1 a_2 \ldots a_n$, we define the reversal of x as: $x^r = a_n a_{n-1} \ldots a_1$.

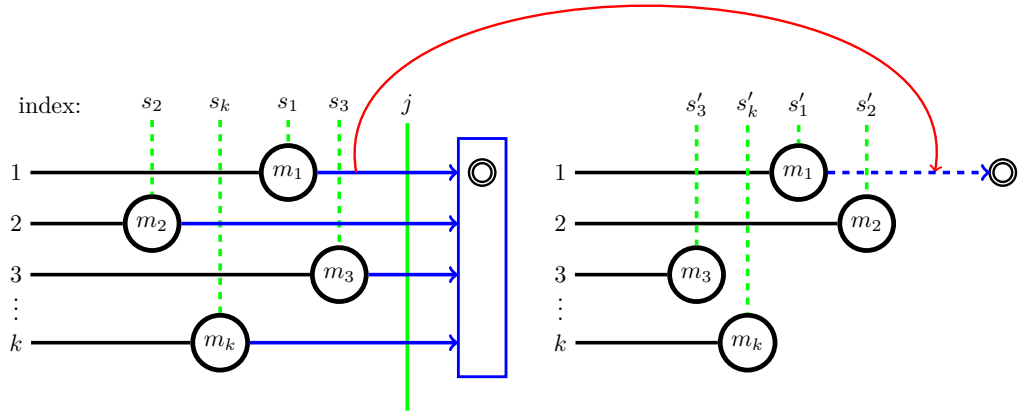


Figure 1 The figure illustrates two k-runs of a k-explorable PDA P. On the left, we depict the k-run $\overline{\rho}_1$, where at a position p an accepting state is reached along the run. On the right, we show $\overline{\rho}_2$, in which an identical modes appears at the same index modulo n, allowing a suffix of ρ_1 (beginning from the first run's step following the matching mode) to be replayed on ρ_2.

We now define VALC$[M]$ as the set of valid computations of M, represented with alternating reversals of configurations, as follows:

$$\{\mathrm{ID}_0(x)\#[\mathrm{ID}_1(x)]^r\#\mathrm{ID}_2(x)\#\cdots\#\mathrm{ID}_{2k}(x) \mid \mathrm{ID}_{2k}(x) \text{ is a halting configuration of } M\}.$$

Each valid computation alternates between standard and reversed configurations, ending in a halting configuration. We also assume that $2k > 3$, that is TM takes at least 3 steps before halting.

We also define the set of invalid computations as: INVALC$[M] = \Gamma^* \setminus$ VALC$[M]$,

INVALC is Explorable

More specifically, in this section we show that INVALC is 2-explorable.

Given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ and an input word $w \in \Sigma^*$, a string α belongs to VALC(M) if and only if the following conditions hold:

1. α is of the form $\alpha = \#C_0\#C_1\#C_2\#\cdots\#C_N\#$ where each C_i is a configuration encoded as a string over $(\Gamma \cup Q)^*$.
2. Each configuration C_i contains exactly one symbol from Q, indicating that the machine is in exactly one state at each computation step.
3. C_0 encodes the initial configuration of M on input w: the tape contains w followed by blanks, the head is at the leftmost position, and the machine is in state q_0.
4. C_N is a halting configuration – i.e., the state in C_N is either q_{accept} or q_{reject}.
5. For every i such that $0 \leq i < N$, the configuration C_{i+1} is a valid successor of C_i under M's transition function δ. And alternate between standard and reversed representations (i.e., C_i is reversed if i is odd).

We observe that among the five conditions defining membership in VALC$[M]$, the first four are *regular*. Therefore, when checking for INVALC$[M]$ only checking (5) – ensuring correct transitions between adjacent configurations – does not hold requires deeper computational power.

Verifying Condition 5 with 2-Explorability

We claim that a 2-explorable PDA suffices to verify that condition (5) does not hold. To do so, we define a 2-explorable machine consisting of a nondeterministic choice between two sub-automata, each of which checks the validity of half the pairs of configurations:

- One sub-automaton processes the sequence in blocks of the form:

  $$\#\underline{C_0\#C_1}\#\underline{C_2\#C_3}\#\cdots\#C_N\#$$

  verifying that each reversed configuration C_{2i+1} is a valid successor of the standard configuration C_{2i}.
- The other sub-automaton processes the alternating pattern:

  $$\#C_0\#\overline{C_1\#C_2}\#\overline{C_3\#C_4}\#\cdots\#C_N\#$$

  verifying that each standard configuration C_{2i+2} is a valid successor of the reversed configuration C_{2i+1}.

Each sub-automaton independently verifies consistency across transitions. Note that checking whether a configuration C' follows validly from C (according to M's transition function) can be done by a DPDA as follows:

- Push the entire configuration C onto the stack while reading.
- Then, read the next configuration C' and pop from the stack, comparing symbols and using the transition function δ to ensure that the head movement, state change, and tape content evolve correctly.
- Any mismatch leads to acceptance

Thus, the only nondeterminism is in the initial choice between the two sub-automata, after which each is deterministic. This means that only two runs suffice for Determiner in the 2-exploration game, and hence the machine 2-explorable.

Since the first four conditions are regular and the fifth is 2-explorable, we conclude that the set of invalid computations $\mathsf{INVALC}[M]$ is recognizable by a 2-explorable PDA.

▶ **Lemma 23** ([6, Lemma 1]). *INVALC[M] is a deterministic context-free language (DCFL) if and only if $L(M)$ is finite.*

Proof. If $L(M)$ is finite, then the set of valid accepting computations VALC[M] is finite and so its complement INVALC[M] is co-finite. Since the class of regular languages is closed under complementation, and all finite or co-finite languages are regular, it follows that INVALC[M] is regular and hence a DCFL.

Conversely, if $L(M)$ is infinite, then VALC[M] is not context-free (since checking the validity of more than two consecutive Turing Machine configurations is not context-free and our machines take at least three steps) and since DCFLs are closed under complement, its complement INVALC[M] cannot be a DCFL. ◀

▶ **Definition 24.** *We say that the relative succinctness of representing deterministic context-free languages (DCFLs) by 2-explorable and deterministic pushdown automata is* not recursively bounded *if there is no recursive function F such that for every 2-explorable-PDA A accepting a DCFL, there exists an equivalent deterministic PDA D such that $|D| \leq F(|A|)$, where $|A|$ and $|D|$ denote the descriptional sizes of A and D, respectively.*

▶ **Theorem 25.** *The relative succinctness of representing deterministic context-free languages by explorable and deterministic PDAs is not recursively bounded.*

Proof. Suppose, for contradiction, that there exists a recursive function F such that for every 2-explorable-PDA A accepting a DCFL, there exists a deterministic PDA D equivalent to A with $|D| \leq F(|A|)$.

Then given a TM M, let A be the 2-explorable PDA that recognizes INVALC[M]. We can compute $F(|A|)$ and enumerate all deterministic PDAs $D_1, D_2, \ldots, D_s$ of sizes up to $F(|A|)$. Then $L(A)$ is not deterministic iff none of these D_j is equivalent to A, which is semi-decidable, by comparing memberships on all strings in Σ^*. If none of them is equivalent to A, we conclude that $L(A)$ is not a DCFL. Therefore, by Lemma 23 we can enumerate the set $\{M \mid L(M) \text{ is infinite }\}$ which would contradict Rice's theorem [12]. Hence, R is not recursively enumerable. Therefore, such a function F cannot exist. ◀

7 Conclusion

In this work, we introduced and studied the notion of *explorability* for PDAs, refining the landscape between deterministic and nondeterministic computation. Building on the concept of history-determinism, we defined the class of k-explorable PDAs and established a strict expressiveness hierarchy based on the number of concurrent explorations permitted.

On the succinctness front, we established that explorable PDAs can be doubly exponentially more succinct than PDAs with fixed constant bound on explorability. Furthermore, the succinctness gap between DPDAs and 2-explorable PDAs is not recursively enumerable, extending the classic non-RE gap between deterministic and nondeterministic PDAs.

Despite these advances, several intriguing questions remain open. The introduction of parameterized explorability gives rise to a rich landscape of intermediate classes – such as $\log n$-explorable, *poly*-explorable or $\sqrt{n}$-explorable PDAs – whose expressiveness and succinctness properties are yet to be systematically studied. Furthermore, while we have established tight bounds in several cases, the exact succinctness gap between general explorable PDAs and fully nondeterministic PDAs remains unresolved. Addressing these questions may not only deepen our understanding of the complexity-theoretic trade-offs between determinism and nondeterminism in the context of PDAs, but could also shed new light on longstanding questions regarding the structure of context-free languages. While we do not have any hope that explorability of PDA is decidable because HD is undecidable, it seems likely that in the context of visibly pushdown automata, explorability might have some better algorithmic properties, like recognizability, lower complexity universality, Gale-Stewart games and inclusion.

References

1 Zuzana Bednárová, Viliam Geffert, Carlo Mereghetti, and Beatrice Palano. Removing nondeterminism in constant height pushdown automata. *Descriptional Complexity of Formal Systems - 14th International Workshop, DCFS 2012, Braga, Portugal, July 23-25, 2012. Proceedings*, 7386:76–88, 2012. doi:10.1007/978-3-642-31623-4_6.

2 Jonathan Goldstine, Hing Leung, and Detlef Wotschke. Measuring nondeterminism in pushdown automata. *STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27 - March 1, 1997, Proceedings*, 1200:295–306, 1997. doi:10.1007/BFB0023467.

3 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:3)2024.

4 Yo-Sub Han, Sang-Ki Ko, and Kai Salomaa. Limited nondeterminism of input-driven pushdown automata: Decidability and complexity. In Michal Hospodár, Galina Jirásková, and Stavros

Konstantinidis, editors, *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17-19, 2019, Proceedings*, volume 11612 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2019. doi:10.1007/978-3-030-23247-4_12.

5 Michael. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1978.

6 Juris Hartmanis. On the succinctness of different representations of languages. *SIAM J. Comput.*, 9(1):114–120, 1980. doi:10.1137/0209010.

7 Emile Hazard, Olivier Idir, and Denis Kuperberg. Explorable parity automata. *CoRR*, abs/2410.23187, 2024. doi:10.48550/arXiv.2410.23187.

8 Emile Hazard and Denis Kuperberg. Explorable automata. *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, 252:24:1–24:18, 2023. doi:10.4230/LIPICS.CSL.2023.24.

9 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.

10 Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity. *Theor. Comput. Sci.*, 181(1):141–157, 1997. doi:10.1016/S0304-3975(96)00267-8.

11 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *ACM SIGACT News*, 32(1):60–65, March 2001. doi:10.1145/568438.568455.

12 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

13 Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of formal languages.* Springer, Berlin, Germany, November 2012.

A Turing Machine

Let M be a Turing machine. Then M is a 7-tuple:$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is a finite set of states,
- Σ is the input alphabet, not containing the blank symbol $\sqcup$,
- Γ is the tape alphabet, where $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{\text{accept}} \in Q$ is the accept state,
- $q_{\text{reject}} \in Q$ is the reject state, with $q_{\text{reject}} \neq q_{\text{accept}}$.

B Pumping Proofs

Ogden's Lemma

Ogden's Lemma is a generalization of the *pumping lemma* for context-free languages. It is particularly useful in proving that certain languages are not context-free.

▶ **Lemma 26** (Ogden's Lemma [5]). *Let L be a context-free language. Then there exists a constant $k \geq 1$ such that for every string $w \in L$ with at least k* marked positions*, there exists a decomposition $w = uvxyz$ such that:*

1. *vxy contains at most k marked positions,*
2. *v and y together contain at least one marked position, and*
3. *for all $i \geq 0$, the string $uv^ixy^iz \in L$.*

▶ **Lemma 10.** *Given $S = (S_n)_{n\in\mathbb{N}}$ where each $S_n \subseteq \mathbb{Z}^+ \times \mathbb{Z}^+$ is finite and non-empty, the language L_S is not context-free.*

Proof. Assume, towards a contradiction, that L'_S is context-free. Then, by Ogden's Lemma as defined in Section 2 , let us choose $w = a^n b^{in} c^{jn}$, where $n > p$, $i, j > 0$, and $(i, j) \in S_n$ are fixed. We mark all n positions in the a-block (i.e., all occurrences of a).

We now consider the possible forms of the substrings v and y. Since only the a-block contains marked positions, at least one of v or y must be a substring of the a-block. Also, in order to preserve the structure of strings in L_S, the decomposition $uvxyz$ must ensure that pumping does not mix symbols from different blocks – otherwise, uv^ixy^iz would no longer be of the form $a^*b^*c^*$, violating the block structure required by L_S.

Thus, v must be entirely within the a-block, and y must lie within a single block – either a, b, or c. In any case, one of the b- or c-blocks will be left untouched by the pumping.

Without loss of generality, suppose y lies in the b-block. Then pumping increases the number of a's (via v) and the number of b's (via y), but leaves the number of c's unchanged. Let us pump $k = jn+1$ times. Since we pumped a non-empty portion of the a-block (required by the Ogdens lemma), the number of a's becomes greater than $jn + 1$, while the number of c's remains exactly jn. This means the resulting string cannot belong to any $L_{i',j'}$ with $i', j' \in \mathbb{N}$, because the ratio of c's to a's is no longer an integer.

Thus, $uv^{jn+1}xy^{jn+1}z \notin L_S$, contradicting Ogden's Lemma. Therefore, L_S is not context-free. ◀

▶ **Lemma 14.** *Given $S' = (S'_w)_{w\in(a^*\#)^{k+1}}$ in which some S'_w are non-empty, then the language $L_{block-S'}$ is not context-free.*

Proof. Assume for contradiction that $L_{\text{block-}S'}$ is context-free. We apply *Ogden's Lemma*.

Let p be the constant from Ogden's Lemma. Define the string:

$$s = a^p \# a^{p+1} \# a^{p+2} \# \cdots \# a^{p+k}.$$

That is, s consists of $k + 1$ distinct a-blocks, each separated by $\#$, and each longer than the previous one. Choose two block lengths i, j occurring in s such that $j > i$, so that $(i, j) \in S'_s$.

Now consider the string:

$$w = s \# b^i c^{j-i} \in L_{\text{block-}S'}.$$

We mark all positions corresponding to the b's in w. There are $i \geq p$ such positions, so Ogden's Lemma applies: there exists a decomposition $w = uvxyz$ such that:

1. vxy contains at most k marked positions,
2. v and y together contain at least one marked position, and
3. for all $i \geq 0$, the string $uv^mxy^mz \in L_{block-S'}$.

Ovserve that if any of v, y cannot contain seperators ($\#$) or 2 distinct letters as it will break the regular strucute of the language.We now consider possible pumping effects:

Case 1: Pumping occurs entirely within the b^ic^{j-i} suffix. If we pump only b's or both b's and c's, we can increase the number of b's and/or c's. However, the prefix s remains unchanged. Since the set S'_s depends only on s, and these are fixed, there is a maximum length $j_{\max}$ of any a-block in s. If $m > p + k$, the pumped string has more b's than any a-block in s, meaning there is no pair $(i', j') \in S'_s$ satisfying the new b/c lengths, contradicting $w_n \in L_{\text{block-}S'}$.

Case 2: Pumping pair v, y lies in an a-block and the b-block. Suppose the pumped region includes some a-block in the prefix s and part of the b-block. Then, pumping increases both the length of that a-block and the number of b's.

We can take $m > p + k$ so that the number of b's becomes larger than all a-blocks in s except possibly the one being pumped. Observe that only one a-block is large, so there is no a block to match the sum of the b and c blocks. This means that the new b-block length must matches the (pumped) a-block's length. In such a situation, the new pair (i', j') extracted from the pumped string, such that i' is the length of a b-block, but as $j' > i'$ but there is no a-block whose length is more than i', the pumped string does not belong to $L_{\text{block-}S'}$. This contradicts Ogden's Lemma.

In all cases, pumping leads to a string not in the language. Thus, our assumption that $L_{\text{block-}S'}$ is context-free must be false. ◀

Synthesising Full-Information Protocols

Dietmar Berwanger
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

Laurent Doyen
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

Thomas Soullard
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, Paris, France

Abstract

We study a communication model where processes reveal their entire local information whenever they interact. However, the system involves an indeterminate environment that may control when a communication event occurs and which participants are involved. As a result, the amount of information a process may receive at once is unbounded.

Such full-information protocols are common in the distributed-computing literature. Here, we consider synchronous systems, modelled as infinite games with imperfect information played on finite graphs. We present a decision procedure for the synthesis of a process with an ω-regular specification in a system where the other participating processes are fixed. The challenge lies in constructing a finite representation of information trees with unbounded branching. Our construction is non-elementary in the size of the problem instance, and we establish a matching non-elementary lower bound for the complexity of the synthesis problem.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Representations of games and their complexity; Computing methodologies → Planning under uncertainty

Keywords and phrases Infinite Games on Finite Graphs, Imperfect Information, Reactive Processes, Distributed Synthesis, Dynamic Networks

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.13

Related Version *Full Version*: `https://arxiv.org/abs/2307.01063`

1 Introduction

One core paradigm for the analysis of complex systems is that of reactive processes, introduced by Harel and Pnueli [17]. A reactive process interacts with its environment in a stepwise, ongoing manner: at each stage, it observes an input signal and chooses a control action with the purpose of ensuring that the system satisfies a specified objective.

Unlike terminating programs that compute a function on a given input, reactive processes are meant to operate indefinitely. Their interaction with the environment is naturally modelled as an infinite-duration game between a strategic player, representing the process, and a non-strategic opponent, Nature. The process aims to satisfy the objective regardless of Nature's choices. The synthesis problem – constructing such a process – thus translates to the construction of a winning strategy in an infinite game [7, 29].

Under perfect information, where input and output uniquely determine the system's run, the game is fully observable and, for ω-regular objectives, it can be solved using established algorithms for parity games [10, 25, 29, 8]. In imperfect-information settings, where processes have only partial views of the system state, the problem becomes significantly harder. Classical approaches, going back to Reif [26, 9], reduce such games to perfect-information ones via a powerset construction, incurring exponential cost but preserving strategy equivalence.


© Dietmar Berwanger, Laurent Doyen, and Thomas Soullard;
licensed under Creative Commons License CC-BY 4.0



In distributed systems, complexity increases further: multiple processes operate based on their local observations and must coordinate to achieve a common objective. This yields coordination games with imperfect information, which are undecidable in general [23, 19, 13]. Even two processes receiving distinct inputs face an undecidable synthesis problem [28]. Coordination may require reasoning about other players' knowledge – a task that is algorithmically intractable over infinite plays [6].

To study distributed synthesis beyond known undecidability barriers, we consider a model where communication is not restricted in content. We formalise a framework of full-information protocols (FIP), inspired by concepts from distributed computing [22, 11, 31], in which all information held by a sender is transmitted during a communication event. Communication availability is controlled externally, and processes cannot choose which information to reveal. This model captures systems with maximal information exchange constrained only by communication opportunities. Whenever synthesis is possible under arbitrary-bandwidth assumptions, it is also possible in the FIP model.

We formalise distributed systems with FIP semantics as repeated games between players and Nature. In each stage, players choose actions; the resulting action profile determines a set of enabled moves, from which Nature selects one. Each move yields an observation profile, including a local input and a set of views shared via communication. A player's view accumulates their own observations and, recursively, those of others with whom they communicates, directly or indirectly. Strategies map such views to actions. A profile of strategies determines a set of plays – infinite sequences of moves – and is winning, if all resulting plays satisfy the objective, expressed by an automaton over infinite words.

As full synthesis remains undecidable in general, we focus on the case of a single active player and an arbitrary number of passive observers. Observers do not act, but their views may be communicated, conveying unbounded information in a single stage. This creates information trees with unbounded branching, obstructing classical synthesis approaches based on finite tree automata [25, 16, 1].

Nevertheless, we prove that synthesis is decidable in this setting. The key lies in a higher-order quotient construction that vastly extends Reif's powerset method to accommodate full-information protocols. It yields a finite game bisimilar to the original one, with winning strategies transferring via a homomorphism. The construction tracks information records along nested coalitions of observers, leading to a state space of non-elementary size.

We show that this complexity is tight: the synthesis problem for FIP games with one active player and n observers is as hard as the acceptance problem for Turing machines using n-fold exponential space. Non-elementary bounds have appeared in similar contexts [2, 19, 13, 14, 5], but are surprising here given the presence of only a single decision maker.

2 Basic Notions

For a function $f : X \to Y$ and a domain subset $Z \subseteq X$, we denote by $f(Z) = \{f(z) \mid z \in Z\}$ the set of images of elements in Z. The *kernel* of f relates elements with the same image $\ker f = \{(x, x') \in X \times X \mid f(x) = f(x')\}$.

For a directed graph (V, E) with edge relation $E \subseteq V \times V$, we designate the set of successors and predecessors of a node $v \in V$ by $vE = \{v' \mid (v, v') \in E\}$ and $Ev = \{v' \mid (v', v) \in E\}$.

For a set of players I, we refer to any nonempty subset $J \subseteq I$ as a *coalition*. A *profile* is a tuple of objects $x = (x^i)_{i \in I}$, one for each player. Given a profile x, we write x^i to designate the element corresponding to Player i. Likewise, for a coalition $J \subseteq I$, we write x^J to designate the profile $(x^j)_{j \in J}$ of objects associated to its members. In general, we use superscripts for (objects associated to) players or coalitions. To avoid confusion, we denote the powerset of a set X by $\wp(X)$ rather than 2^X.

For an alphabet Γ, the set of finite words over Γ is denoted by Γ^*, the set of finite nonempty words is $\Gamma^+ = \Gamma^* \setminus \{\varepsilon\}$, and the set of infinite words is Γ^ω.

2.1 Games on Graphs

In the context of reactive systems, games are used to study worst-case scenarios where a system interacts with an adversarial environment. The goal is to synthesise a strategy for the system player that ensures the specification is satisfied regardless of the environment's behaviour. Accordingly, we focus on actions and strategies of one player – or a coalition of players – representing the system, and attribute the environment choices to a nonstrategic player, Nature. In particular, we deviate from traditional terminology by featuring two-player games as games between one player and Nature. To compare games played on different structures, we represent objectives in terms of colours assigned to game positions.

Perfect Information

Let A be a finite set of actions and C a finite set of colours. A *graph game with perfect information* is described by a coloured graph $\mathcal{G} = (V, v_\varepsilon, (E_a)_{a\in A}, \lambda)$, where V is a set of positions, $v_\varepsilon \in V$ the initial position, each $E_a \subseteq V \times V$ is a transition relation for action a, and $\lambda\colon V \to C$ labels positions with colours. For all $v \in V$ and $a \in A$, we assume that the successor set vE_a is nonempty.

The game is played in stages starting from the initial position v_ε. In a stage at position v, the player chooses an action $a \in A$, then Nature selects an edge $(v, w) \in E_a$, and the play moves to position w, which is announced to the player. The outcome is an infinite path in $\mathcal{G}$ starting from v_ε, called a *play*; finite prefixes are called *histories*.

A reachability objective is given by a binary labelling $\lambda\colon V \to \{0, 1\}$, and a play is winning if it visits a position labelled 1. A parity objective is specified by a labelling $\lambda\colon V \to \mathbb{N}$ that assigns priorities to positions, and a play is winning if the least priority visited infinitely often is even.

Strategies in games with perfect information are maps $s\colon V^* \to A$ from histories to actions. The *outcome* of a strategy s is the set $\mathrm{Out}(s)$ of plays $v_0v_1v_2\dots$ such that $v_{t+1} \in v_tE_a$ for $a = s(v_0v_1\dots v_t)$ at every stage $t \geq 0$. A strategy is winning, if all the plays in $\mathrm{Out}(s)$ are winning. The synthesis problem asks whether such a strategy exists and aims to construct one effectively.

For parity and reachability objectives, it is known that whenever a winning strategy exists, there exists one that depends only on the last position in the history. Such positional strategies can be represented by a labelling of positions with actions. As the winning status of a positional strategy can be verified efficiently, the synthesis problem for parity games is decidable in $\mathrm{NP} \cap \mathrm{CoNP}$; reachability games are solvable in polynomial time.

Partial Observation

One way to model uncertainty is by associating to each position an observation from a finite alphabet B, via a function $\beta\colon V \to B$. In a stage where the game moves to position w, only $\beta(w)$ is revealed to the player.

Formally, a *graph game with partial observation* is given by a coloured graph $(V, v_\varepsilon, (E_a)_{a\in A}, \beta, \lambda)$. Every history $\tau = v_0v_1\dots v_t$ yields an observation history $\hat{\beta}(\tau) := \beta(v_0)\beta(v_1)\dots\beta(v_t)$. Strategies are functions $s\colon V^* \to A$ from histories to actions that do not distinguish between histories with the same observation. Under the assumption that objectives are *visible*, in the sense that indistinguishable histories end in positions with the

same colour given by λ, the synthesis problem for partial-observation games on graphs with parity or reachability objectives can be reduced to a perfect-information game via a powerset construction, yielding an EXPTIME-complete problem [26, 9].

To model distributed systems, graph games with partial observation are extended to multiple players $i \in I$, each with an action set A^i, an observation set B^i, and a local observation function $\beta^i \colon V \to B^i$. The edges are then indexed by action profiles $a = (a^i)_{i \in I}$. At the stage game from position v, each player i selects an action $a^i \in A^i$, Nature chooses an edge $(v, w) \in E_a$, and each player observes $\beta^i(w)$. Strategies s^i of Player i are functions that do not distinguish between her observation histories. The outcome of a strategy profile is formed by the plays that are in the outcome of all component strategies.

The synthesis problem for a given, common objective asks whether there exists a strategy profile $s = (s^i)_{i \in I}$, such that all outcoming plays satisfy the objective. This problem is undecidable in general for distributed games with partial observation, even with only two players against Nature (see, e.g., [28]).

2.2 Automata

We use deterministic finite automata as language acceptors for infinite words and as transducers on finite words.

A *Mealy automaton* is given by a tuple $(Q, \Gamma, \Sigma, q_\varepsilon, \delta, \lambda)$ consisting of a finite set Q of states, a finite input alphabet Γ, a finite output alphabet Σ, a designated initial state $q_\varepsilon \in Q$, a transition function $\delta : Q \times \Gamma \to Q$, and an output function $\lambda : Q \times \Gamma \to \Sigma$. To describe the internal behaviour, we extend the transition function from input letters to words and define $\delta \colon Q \times \Gamma^* \to Q$, for every state $q \in Q$, by setting $\delta(q, \varepsilon) := q$, for the empty word ε, and $\delta(q, \tau c) := \delta(\delta(q, \tau), c)$, for words $\tau \in \Gamma^*$ and letters $c \in \Gamma$. Likewise, to describe the external behaviour, we extend the output function to $\lambda : \Gamma^+ \to \Sigma$ by setting $\lambda(\tau c) = \lambda(\delta(q_\varepsilon, \tau), c)$ for all $\tau \in \Gamma^*$ and $c \in \Gamma$. A function on Γ^* is *regular* if there exists a Mealy automaton that defines it. Further, we define the *cumulative output*, for an input word $\tau = c_1 c_2 \dots c_n \in \Gamma^*$, as the sequence $\hat{\lambda}(\tau) = \lambda(c_1)\lambda(c_1 c_2) \dots \lambda(c_1 c_2 \dots c_n)$ consisting of the outputs of all prefixes of τ. Finally, we extend $\hat{\lambda}$ to infinite words $\pi = c_1 c_2 \dots \in \Gamma^\omega$, by setting $\hat{\lambda}(\pi) = \lambda(c_1)\lambda(c_1 c_2) \dots$

3 The Model

3.1 Repeated Games with Imperfect Information

Our purpose is to model dynamical systems driven by occurrences of discrete state changes that we call *moves*. System runs correspond to infinite sequences of moves drawn from a finite set Γ as outcome of a multistage game played between a fixed set $I = \{0, 1, \dots, n\}$ of players and Nature (or Environment, in control-theoretic terminology). Each player $i \in I$ has a set A^i of actions, and any action profile $(a^i)_{i \in I}$ enables a set of moves, according to a move-action map $\mathrm{act} \colon \Gamma \to A$, which is *surjective*. In every stage, a one-shot *base game* is played as follows: each player $i \in I$ chooses an *action* a^i from her given action set A^i; the chosen profile $a = (a^i)_{i \in I}$ constrains the set of possible outcomes to the subset of moves $\{c \in \Gamma \mid \mathrm{act}(c) = a\}$ supported by a, from which Nature chooses one. The outcoming move is recorded in the play history, and the game proceeds to the next stage. The outcome of the multistage game, called a *play*, is thus an infinite sequence $\pi = c_1 c_2 \dots \in \Gamma^\omega$ of moves. A *history* (of length ℓ) is a finite prefix $\tau = c_1 c_2 \dots c_\ell \in \Gamma^*$ of a play; the empty history ε has length zero. The objective of a player is described by a subset of plays declared as winning.

To pursue their objectives, players choose actions based on the information available to them. The information of a player $i \in I$ is modelled by a partition $\mathcal{U}^i$ of the set Γ^* of histories; the parts of $\mathcal{U}^i$ are called *information sets* (of the player). The intended meaning is that if the actual history belongs to an information set $U \in \mathcal{U}^i$, then Player i considers every history in U possible. The particular case where all information sets in the partition are singletons characterises the setting of *perfect information*.

Our model is *synchronous*, which means, intuitively, that players always know how many stages have been played. Formally, this amounts to asserting that all histories in an information set have the same length; in particular the empty history forms a singleton information set. Further, we assume that every player has *perfect recall* – he never forgets what he knew previously, and which actions he took. Formally, if an information set of Player i contains nontrivial histories τc and $\tau' c'$, then the predecessor history τ is in the same information set as τ' and the moves c and c' are supported by the same action of Player i.

A *decision function* for a player $i \in I$ is a map $f^i \colon \Gamma^* \to A^i$ from histories to actions. We say that a play, or a history, $c_1 c_2 \dots$ *follows* f^i if $\mathrm{act}^i(c_t) = f^i(c_1 \dots c_{t-1})$, for every stage $t > 0$. Further, we say that an information set $U \in \mathcal{U}^i$ is *reachable* if there exists a history in U that follows f^i. A decision function f^i is *information consistent* if it is constant on all reachable information sets. A *strategy* for Player i is a decision function that is information consistent (with respect to her information partition). The outcome of a strategy f^i is the set $\mathrm{Out}(f^i) \subseteq \Gamma^\omega$ of plays that follow it. To capture coordination problems in distributed systems, we refer to an objective $W \subseteq \Gamma^\omega$ that is common to all players. A strategy profile $f = (f^i)_{i \in I}$ is *winning* if $\mathrm{Out}(f) \subseteq W$.

Finite-state representation

As we are interested in algorithmic questions for repeated games, we will consider instances described by finite objects. We assume that the move set Γ and the set A of action profiles are finite. Hence, the action map $\mathrm{act} \colon \Gamma \to A$ provides a finite description of the move structure, that is, Γ^* equipped with the labelling of action profiles.

To represent objectives – sets of infinite words from Γ^ω – we use colouring functions defined by Mealy automata over Γ. We mainly work with *parity* automata, which provide a canonical form for ω-regular specifications relevant as system-design objectives [30]. These are automata that input moves – drawn during the play – and output natural numbers called *priorities*: a play is winning if the least priority output infinitely often is even. We sometimes refer to reachability objectives, given by an automaton that outputs flags 0 or 1: a play is winning if some prefix history yields output 1.

To describe information partitions in a finite way, it will be convenient to refer to their representation as *indistinguishability relations*, which equate histories that the player cannot distinguish. An information partition $\mathcal{U}$ thus corresponds to the equivalence relation $\sim \in \Gamma^* \times \Gamma^*$ with $\tau \sim \tau'$ whenever $\tau, \tau' \in U$ for some $U \in \mathcal{U}$. Generally, an indistinguishability relation for Player i is a synchronous equivalence $\sim$ on $(\Gamma \times \Gamma)^*$ that is prefix-closed and respects the player's actions: if $\tau c \sim \tau' c'$ then $\tau \sim \tau'$ and $\mathrm{act}^i(c) = \mathrm{act}^i(c')$, for all $\tau, \tau' \in \Gamma^*$ and $c, c' \in \Gamma$. Indeed, $\sim$ defines an information partition with information sets given by equivalence classes $[\tau]_\sim = \{\tau' \in \Gamma^* \mid \tau' \sim \tau\}$, for each history $\tau \in \Gamma^*$. The information structures arising from the full-information protocols introduced in this paper are a particular case of indistinguishability relations recognisable by two-tape deterministic finite automata, as studied in [4].

We shall distinguish between the finite-state representation of a repeated game, for instance as a tuple $(\mathrm{act}, (\mathcal{R}^i)_{i \in I}, \mathcal{A})$, on the one hand, that includes automata $\mathcal{R}^i$ and $\mathcal{A}$ describing the indistinguishability relations and the objective, and its presentation as a logical structure $\mathcal{G} = (\Gamma^*, \mathrm{act}, (\sim^i)_{i \in I}, \lambda)$, on the other hand, with $\sim^i = L(\mathcal{R}^i)$ and $\lambda = \lambda^{\mathcal{A}}$.

3.2 Full-Information Protocols

We formalise a communication model that gives rise to a particular class of finitely-representable information structures for repeated games. Our formalisation and the application scenario are inspired from [21].

In full-information protocols, players receive local observations from a finite set B^i via an observation function β^i, as they do in partial-observation games, but additionally they communicate with other players. The information transfer in that event is idealistically efficient: the entire information available to the sending party is revealed to the receiving player. However, the opportunity of communication may not be under the control of the players: whether a communication event occurs in a particular stage, and which player it includes, is determined by the current move.

Communication opportunities for a player $i \in I$ are specified by a function $\mathrm{com}^i \colon B^i \to \wp(I)$ that associates with each of her local observation $b \in B^i$, a set of players $\mathrm{com}^i(b^i) \subseteq I$ to which communication links are enabled. Intuitively, when Player i receives the local observation $b^i \in B^i$, she also receives the information of every player $j \in \mathrm{com}^i(b^i)$, which includes his observation history $b^j_1, b^j_2, \ldots, b^j_\ell$, but also (recursively) the observation history of all players in $\mathrm{com}^j(b^j_t)$ at previous stages $t = 1, 2, \ldots, \ell$. Thus, a link $j \in \mathrm{com}^i(b^i)$ specifies a one-way communication event from sender j to receiver i: upon observing b^i, Player i can peek at Player j. We refer to such links as *direct* links. Our semantics of communication links is transitive. If at some history, there is a direct link from Player i to Player j, and also a direct link from Player j to Player k, then an *indirect* communication link is established from Player i to Player k. Even if the protocol specifies no direct link from i to k, the information of k is revealed to i.

Formally, we represent the information available to the players at a history $\tau = c_1 c_2 \ldots c_\ell$ by a *view graph* $\mathrm{View}(\tau) = (V, E)$, defined as follows:

- $V = I \times \{0, 1, \ldots, \ell\}$ is the set of nodes, and a node $(i, t) \in V$ represents the viewpoint of Player i in stage t;
- $E \subseteq V \times V$ is the set of edges, where an edge $((i, t), (j, u))$ intuitively means that in stage t, Player i has access to the view of Player j in stage u; the set E contains the edges $((i, t), (i, t-1))$ for all stages $1 < t \leq \ell$ and every player i – which correspond to looking into the past, and the edges $((i, t), (j, t))$ are included, for all stages $1 \leq t \leq \ell$ and all players $i \in I$ and $j \in \mathrm{com}^i(b^i)$ where $b = \beta^i(c_1 c_2 \ldots c_t)$ – which correspond to revealing the current view of Player j to Player i (via a direct link).

Two histories $\tau, \tau' \in \Gamma^*$ are indistinguishable for Player i, denoted $\tau \sim^i \tau'$, if they have the same length ℓ and the local observations $\beta^j(\tau(t)) = \beta^j(\tau'(t))$ agree, for all nodes (j, t) reachable from (i, ℓ) in the view graph $\mathrm{View}(\tau)$. In particular, this means that from (i, ℓ) the same set of nodes is reachable in $\mathrm{View}(\tau)$ and in $\mathrm{View}(\tau')$. We say that the histories τ, τ' are indistinguishable for a coalition $J \subseteq I$, denoted $\tau \sim^J \tau'$, if they are indistinguishable for all players of the coalition, that is, $\tau \sim^i \tau'$ for all $i \in J$.

A *full-information protocol* (FIP) for a set of players I, a move alphabet Γ, and an alphabet B of observation profiles, is described by a profile $F = (\mathcal{M}^i, \mathrm{com}^i)_{i \in I}$ specifying for each player $i \in I$, a Mealy automaton $\mathcal{M}^i$ that defines the local observation function $\beta^i \colon \Gamma^* \to B^i$ and the communication map $\mathrm{com}^i \colon B^i \to \wp(I)$ that specifies, for every local observation, the set of players to which a communication link from i is enabled. The protocol F defines an indistinguishability relation, for each player $i \in I$, which we denote by $\sim^i_F$.

To turn a FIP instance $F = (\mathcal{M}^i, \mathrm{com}^i)_{i \in I}$ into a game, consider now a profile A of action sets A^i for the players $i \in I$ together with a suitable action map $\mathrm{act} \colon \Gamma \to A$. The actions of any player i should be observable to himself: if $\mathrm{act}^i(c) \neq \mathrm{act}^i(c')$, then $\beta^i(\tau c) \neq \beta^i(\tau' c')$,

for all histories $\tau, \tau' \in \Gamma^*$ and moves $c, c' \in \Gamma$. If the condition is satisfied, then F induces a profile of indistinguishability relations for the repeated game form described by act. Together with the action map act and a parity automaton $\mathcal{A}$ for defining the objective, we thus obtain the repeated game $\mathcal{G}(F) := (\Gamma^*, \mathrm{act}, (\sim_F^i)_{i\in I}, \lambda^{\mathcal{A}})$, which we call the *FIP game* associated with F.

3.3 Example: Leader election with failures and recoveries

To illustrate the model, consider a classical leader election scenario. A set $V = \{0, \ldots, n\}$ of processes is connected in a network graph (V, E), where each edge $(i, j) \in E$ represents a unidirectional channel through which process i can send messages to process j. The system runs indefinitely in synchronous rounds.

In each round $t \in \mathbb{N}$, every process i declares a leader candidate $a_t^i \in V$ and proposes a message m_t^i for broadcasting to its neighbours $j \in iE$. The environment may crash a subset $F_t \subseteq E$ of channels, so that process i receives messages only from $j \in (E \setminus F_t)i$. In the next round, some channels may recover while others may fail. A process i is said to fail at time t if any of its outgoing channels crash. Failures are bounded by parameters (N, L): at most N channels may crash in any window of L rounds. A process is valid at time t if it does not fail.

The objective of leader election is to ensure that, unless all processes fail repeatedly, there is a time t_0 after which all valid processes i agree on a leader $a_t^i = k$ who is also valid.

Note that we did not specify the message space. This is intentional: in practice, it is often difficult to settle on the structure of internal messages for distributed algorithms that are yet to be designed. To maintain focus on external behaviour, we instead assume that processes broadcast all available information whenever communication is possible – in other words, they follow a full-information protocol. Once a solution is found for this idealised setting, it may later be implemented more efficiently. It might turn out that the problem is unsolvable even under full information, hence no effective protocol exists either.

Process implementation must answer two questions: (1) How to choose a leader based on the observed failure history; (2) What to transmit. The full-information assumption focuses on the first question, assuming the second is solved or deferred.

We model this as a FIP game with players in $I = V \cup E$. Each process player $i \in V$ has actions $A^i = I$, representing candidates, whereas channel players are passive, with $|A^e| = 1$, for all $e \in E$. Moves are tuples (a, F) of declarations and crash sets, with $\Gamma = \prod_{i\in V} V \times \wp(E)$ and $\mathrm{act}(a, F) = a$.

Local observations, for each process player i, specify the subset of channels from which it can read, i.e., $B^i \subseteq \{(j, i) \in E\}$, and the communication function com^i is the identity. Including the channels as observers is an artifice to prevent players from seeing the entire network, since we choose a semantics where communication is transitive. Accordingly, each channel (i, j) has two observations: $\{\mathrm{read}, \mathrm{lock}\}$ with $\mathrm{com}^{(i,j)}(\mathrm{read}) = \{i\}$ – when the source process is revealed – and $\mathrm{com}^{(i,j)}(\mathrm{lock}) = \varnothing$ – when it may reveal to the target. The observation automata for channels alternate between read at odd times and lock at even times. For processes, the observation automaton of i sends the $\varnothing$ observation at odd times, whereas it reveals the enabled incoming channels from $E \setminus F$ at even times. The Mealy automaton is further adjusted to keep track of the number of crashes within the time window. To ensure that the bounds (N, L) are respected, the automata send $\varnothing$ when the budget does not allow to crash the subset F recorded in the current move.

The objective is described by a parity automaton with three priorities: 2 when all valid processes agree on a valid leader; 1 when there is disagreement or the leader is invalid; 0 after a phase in which all processes failed. It rejects plays that visit priority 1 infinitely often.

Now we can verify that the model faithfully captures our scenario: any distributed protocol – regardless of which messages the processes pass – corresponds to a strategy profile. It solves the leader election problem under the given bounds if, and only if, the strategy is winning in the FIP game.

4 The Synthesis Problem

We study the following synthesis problem: Given a finite FIP game instance $\mathcal{F} = (\mathrm{act}, (\mathcal{M}^i, \mathrm{com}^i)_{i\in I}, \mathcal{A})$, decide whether there exists a winning strategy in the repeated game $\mathcal{G}(\mathcal{F})$ and, if so, effectively construct one.

Graph games are special cases of FIP games. For instance, a perfect-information game on a graph $(V, v_\varepsilon, (E_a)_{a\in A}, \lambda)$ corresponds to the repeated game $(\mathrm{act}, \mathrm{id}, \mathcal{M})$ with $\Gamma = A \times V$, where $\mathrm{act}(a, v) = a$ and $\mathcal{M}$ maps a history $\tau = (a_1, v_1)\dots(a_t, v_t)$ to $\lambda(v_t)$ if τ follows a path from v_ε, and to a fresh colour $\perp$ otherwise. Given an objective $L \subseteq C^\omega$, a play is winning in the repeated game, if $\mathcal{M}$ maps it to a sequence in L or to one reaching $\perp$.

Partial-observation games translate similarly by expressing observation functions as Mealy automata that output observation labels on transitions. Standard partial-observation games [26] correspond precisely to FIP games with one player. In such repeated games, the indistinguishability relation $\sim$ is defined by a regular observation function $\beta : \Gamma^* \to B$, such that $\tau \sim \tau'$ if $\hat{\beta}(\tau) = \hat{\beta}(\tau')$. Assuming perfect recall, $\tau c \sim \tau' c'$ iff $\tau \sim \tau'$ and $\beta(\tau c) = \beta(\tau' c')$. Accordingly, every information set U has at most $|B|$ successors, so the information tree has bounded branching.

For two or more players, FIP games yield strictly richer information structures. Consider a FIP game with players $I = \{0, 1\}$, binary observations, and a communication map where Player 0 can communicate with Player 1 only on observation 1. Suppose Player 0 observes 0 for 7 stages, while Player 1 observes an arbitrary bit sequence. In stage 8, Player 0 observes 1, revealing all 8 bits seen by Player 1. Thus, her current information set, with 2^7 histories, has 2^8 successors (that are singletons). If the play continues in the same way, until the next communication event occurs, say in stage 88, there will be a branching of degree 2^{80}, and so on. Hence, FIP information trees may have unbounded branching.

Restriction to one active player, visible objectives

Since synthesis is undecidable already for two players in partial-observation games, we restrict our analysis to one active player. Accordingly, we consider games with a set of players $I = \{0, \dots, n\}$, where Player 0 is active and the others are passive observers ($|A^i| = 1$ for $i \neq 0$). References to actions or indistinguishability relations refer to Player 0 unless specified.

We further restrict to *visible* objectives, where the colouring function $\lambda^{\mathcal{A}}$ defined by the objective automaton $\mathcal{A}$ is information consistent: $\lambda^{\mathcal{A}}(\tau) = \lambda^{\mathcal{A}}(\tau')$ whenever $\tau \sim^0 \tau'$.

Despite these restrictions, the model captures significant system-design problems. One motivation is monitoring and diagnosis: constructing a decentralised supervisor that runs alongside a fixed distributed system – using the given communication architecture – to log data and detect faults. The active player outputs new data according to the specified objective, leaving the system unchanged.

An active player added to a fixed distributed protocol can influence the system meaningfully. For example, in leader election, we may let one process repair channels in addition to its original role, with the objective to ensure stability, that is, that leaders are revoked only if they fail. Formally this is done by extending Player 0's action set to $A^0 \times E$, and adapting the action map to exclude the repaired channel from the crash set of the stage outcome. The

synthesis objective is stability in conjunction with the condition that passive players follow the actions prescribed by the original protocol. Under standard connectivity assumptions (already necessary to solve basic leader election), this can be specified by a visible automaton.

4.1 Method

In the rest of the paper, we show that the synthesis problem for FIP games with one active player is computationally solvable, albeit with daunting theoretical complexity.

The key tool for synthesis in infinite games is the automata-theoretic approach, founded by Büchi and Landweber [7] and Rabin [24, 25]. Strategies are viewed as trees with bounded branching. However, FIP games may yield unbounded information trees. Indeed, [4] shows that a regular indistinguishability relation defines a tree of finite branching degree if, and only if, it corresponds to an observation function. The counterexample there is a FIP protocol with one active player and one passive observer. Thus, we cannot rely on tree automata to recognise the set of winning strategies of a FIP game.

To overcome this obstacle, we reduce a finite FIP game (with parity objectives) to a perfect-information graph game, preserving winning strategies: (1) if a winning strategy exists in the original game, one exists in the reduct; (2) from a regular strategy in the reduct, we can construct one for the original.

The reduction builds effectively a homomorphism from histories Γ^* to a finite set of abstract states, preserving information sets. By collapsing the abstract states of an information set, we obtain a finite perfect-information game equivalent (bisimilar) to the original. This allows pulling back winning strategies.

Information quotient

A repeated game with imperfect information and one active player can be represented as a game with perfect information played on the information tree – the infinite graph with information sets as positions.

Let us fix a move alphabet Γ and a set A of actions. The *information quotient* of a game $\mathcal{G} = (\Gamma^*, \mathrm{act}, \sim, \lambda)$ is the graph $\mathcal{U}(\mathcal{G})$ on $U = \{[\tau]_\sim \mid \tau \in \Gamma^*\}$, with initial node $u_\varepsilon = [\varepsilon]_\sim$, edges:

$$E_a^{\mathcal{U}} := \{([\tau]_\sim, [\tau c]_\sim) \mid \mathrm{act}(c) = a\}, \quad \text{for each } a \in A,$$

and colouring $\lambda^{\mathcal{U}}([\tau]_\sim) = \lambda(\tau)$. As $\sim$ has perfect recall, $\mathcal{U}(\mathcal{G})$ is a tree. Therefore, each node $u \in U$ identifies a unique path from u_ε, and we view strategies as functions $s\colon U \to A$.

Although structurally different, a game with imperfect information and the perfect-information game played on its information tree are the same (from the perspective of the active player), in the following sense.

▶ **Lemma 1.** *For every game $\mathcal{G}$ with indistinguishability relation $\sim$, there is a one-to-one correspondence mapping each strategy s in $\mathcal{U}(\mathcal{G})$ to a strategy $\tilde{s}$ in $\mathcal{G}$, such that $\tilde{s}(\tau) = s([\tau]_\sim)$, and their outcomes agree on colours: $\hat{\lambda}(\mathrm{Out}(\tilde{s})) = \hat{\lambda}^{\mathcal{U}}(\mathrm{Out}(s))$.*

Bisimulation

Further, we use bisimulation to relate strategies across games.

Let $\mathcal{G}$, $\mathcal{H}$ be perfect-information game graphs with the same vocabulary $(v_\varepsilon, (E_a)_{a\in A}, \lambda)$. A *bisimulation* between $\mathcal{G}$ and $\mathcal{H}$ is a relation $Z \subseteq V^{\mathcal{G}} \times V^{\mathcal{H}}$ such that every related pair of nodes $(u, v) \in Z$ agree on the colour $\lambda^{\mathcal{G}}(u) = \lambda^{\mathcal{H}}(v)$ and

(Zig) for each $a \in A$ and $(u, u') \in E_a^{\mathcal{G}}$, there is $(v, v') \in E_a^{\mathcal{H}}$ with $(u', v') \in Z$;
(Zag) for each $a \in A$ and $(v, v') \in E_a^{\mathcal{H}}$, there is $(u, u') \in E_a^{\mathcal{G}}$ with $(u', v') \in Z$.
The graphs $\mathcal{G}$, $\mathcal{H}$ are *bisimilar* if their initial states are related: $(v_\varepsilon^{\mathcal{G}}, v_\varepsilon^{\mathcal{H}}) \in Z$.

A graph *homomorphism* from $\mathcal{G}$ to $\mathcal{H}$ is a function $h\colon V^{\mathcal{G}} \to V^{\mathcal{H}}$ that preserves the initial state: $h(v_\varepsilon^{\mathcal{G}}) = v_\varepsilon^{\mathcal{H}}$, the edges: $h(E_a^{\mathcal{G}}) \subseteq E_a^{\mathcal{H}}$ for all $a \in A$, and the colouring: $\lambda^{\mathcal{G}} = \lambda^{\mathcal{H}} \circ h$. A *p-morphism* is a homomorphism $h\colon \mathcal{G} \to \mathcal{H}$ such that the relation $\{(x, h(x)) \mid x \in V^{\mathcal{G}}\}$ forms a bisimulation. Equivalently, h is a homomorphism and, for every edge $(v, w) \in E_a^{\mathcal{H}}$ and node $x \in V^{\mathcal{G}}$ with $h(x) = v$, there exists an edge $(x, y) \in E_a^{\mathcal{G}}$ such that $h(y) = w$ [15, 27].

▶ **Lemma 2** ([12]). *Let $\mathcal{G}$ and $\mathcal{H}$ be bisimilar game graphs. Then, $\mathcal{G}$ has a winning strategy iff $\mathcal{H}$ has one. Moreover, if there exists a p-morphism $h\colon \mathcal{G} \to \mathcal{H}$, then it takes every strategy s in $\mathcal{H}$, to a strategy $s \circ h$ in $\mathcal{G}$ such that the outcomes agree on colours: $\hat{\lambda}^{\mathcal{G}}(\mathrm{Out}(s \circ h)) = \hat{\lambda}^{\mathcal{H}}(\mathrm{Out}(s))$.*

5 The Reduction

For this section, let us fix a set $I = \{0, \dots, n\}$ of players with 0 being the active player, with an action set A, and the others being passive observers.

5.1 Simplifications

To simplify the presentation, we assume that all players in I share a common observation alphabet B. Moreover, we identify moves with observation profiles: $\Gamma = \prod_{i \in I} B$. Each player's observation function is defined trivially by $\beta^i(\tau c) = c^i$ for all $\tau \in \Gamma^*$ and $c \in \Gamma$.

These assumptions can be made without loss of generality. Any FIP instance can be brought to this form by adding a dummy observer and modifying the objective to check that the observations in the view of the active players match the original Mealy-automaton outputs, or else go to a sink state labelled with an even priority. This modification incurs an exponential blowup in size, but preserves visibility of the objective.

In the simplified model, we can characterise indistinguishability relations without reference to view graphs. Given a move $c \in \Gamma$, let $\mathrm{Link}(c) := \bigcup_{i \in I} \{i\} \times \mathrm{com}^i(c^i)$ denote the set of all communication links enabled under c. For any player $i \in I$, let now $\mathrm{sync}^i(c)$ be the set of players reachable from i via the reflexive-transitive closure of $\mathrm{Link}(c)$. Hence, $\mathrm{sync}^i(c)$ describes the set of players that communicate (directly or indirectly) with Player i under move c. For a coalition $J \subseteq I$, let $\mathrm{sync}^J(c) := \bigcup_{i \in J} \mathrm{sync}^i(c)$.

We note the following properties for later use: $\mathrm{sync}^J(c)$ always includes J, every coalition $L = \mathrm{sync}^J(c)$ that occurs as a synchronisation target is *autonomous*, meaning $L = \mathrm{sync}^L(c)$, intermediary coalitions $J \subseteq K \subseteq \mathrm{sync}^J(c)$ share the same target: $\mathrm{sync}^K(c) = \mathrm{sync}^J(c)$, and $\mathrm{sync}^{J \cup K}(c) = \mathrm{sync}^J(c) \cup \mathrm{sync}^K(c)$.

The following lemma describes how the information sets of a coalition evolve with a move.

▶ **Lemma 3.** *Let $J \subseteq I$ be a coalition. Then, for all histories $\tau \in \Gamma^*$ and all moves $c \in \Gamma$:*

(i) *For the target coalition $L = \mathrm{sync}^J(c)$, we have $[\tau c]_{\sim^L} = \{\tau' d \mid \tau' \sim^L \tau \text{ and } d^L = c^L\}$.*
(ii) *For every intermediary coalition K with $J \subseteq K \subseteq \mathrm{sync}^J(c)$, we have $[\tau c]_{\sim^K} = [\tau c]_{\sim^J}$.*

Lemma 3 provides an inductive definition of indistinguishability in simplified FIP games: $[\tau c]_{\sim^J} = [\tau c]_{\sim^L} = \{\tau' d \mid \tau' \in [\tau]_{\sim^L} \text{ and } d^L = c^L\}$, for $L = \mathrm{sync}^J(c)$. Recall that, for the grand coalition, $\sim^I$ is the identity.

5.2 Abstraction morphism

As a first step towards our reduction, we map each history $\tau \in \Gamma^*$ to an abstract state from a finite domain. This state encodes the knowledge of every coalition containing the active player 0. For a coalition $J \subseteq I$, its own knowledge is information-consistent, hence visible; whereas knowledge of strictly larger coalitions $K \supsetneq J$ might not be information-consistent for coalition J – these values form a hidden part of the abstract state, called the configuration.

Knowledge is represented by a set of configurations, and configurations are profiles of knowledge sets indexed by coalitions. For the grand coalition, the configuration is just the state of the objective automaton.

For any $J \subseteq I$, let $J^{\uparrow} := \{K \subseteq I \mid J \subseteq K\}$ be the set of coalitions containing J, and $J^{\Uparrow} := J^{\uparrow} \setminus \{J\}$. We call any $J \in \{0\}^{\uparrow}$ an *active coalition*.

Let $\mathcal{F} = (\mathrm{act}, (\mathrm{sync}^J)_{J \in \{0\}^{\uparrow}}, \mathcal{A})$ be a FIP game in simplified form, with local observation alphabet B, moves Γ, action map $\mathrm{act}\colon \Gamma \to A$, a profile of synchronisation maps $\mathrm{sync}^J\colon \Gamma \to \wp(I)$, one for every active coalition J, and a Mealy automaton $\mathcal{A}$ defining the objective. We refer to the corresponding one-player game as $\mathcal{G} := (\Gamma^*, \mathrm{act}, \sim, \lambda^{\mathcal{A}})$.

For each $J \in \{0\}^{\uparrow}$ and history $\tau \in \Gamma^*$, we define the *configuration* $\varphi_J(\tau)$ and the *knowledge set* $\Phi^J(\tau)$ by mutual induction:

- $\varphi_I(\tau) := \delta^{\mathcal{A}}(q_\varepsilon, \tau)$,
- $\varphi_J(\tau) := (\Phi^K(\tau))_{K \in J^{\Uparrow}}$ for $J \neq I$,
- $\Phi^J(\tau) := \{\varphi_J(\tau') \mid \tau' \sim^J \tau\}$.

Finally, the *abstract state* for coalition J is $h_J(\tau) := (\Phi^J(\tau), \varphi_J(\tau))$. We write Conf_J and KSet^J for the ranges of φ_J and Φ^J, respectively. These sets are finite, as $\mathrm{Conf}_I = Q$ and $\mathrm{Conf}_J \subseteq \prod_{K \in J^{\Uparrow}} \mathrm{KSet}^K$, whereas $\mathrm{KSet}^J \subseteq \wp(\mathrm{Conf}_J)$. The set of abstract states is $Q^{\mathcal{H}}_J \subseteq \mathrm{KSet}^J \times \mathrm{Conf}_J$. We omit indices for $J = \{0\}$ and write $h = h_{\{0\}}$.

Note that in any $(\Psi, \psi) \in Q^{\mathcal{H}}_J$, we have $\psi \in \Psi$. Also, the knowledge-set component of h_J is information-consistent by construction.

▶ **Lemma 4** (Preservation of indistinguishability). *If $\tau \sim^J \tau'$, then $\Phi^J(\tau) = \Phi^J(\tau')$, for all histories $\tau, \tau' \in \Gamma^*$ and all active coalitions $J \in \{0\}^{\uparrow}$.*

In particular, this holds for the active player singleton $J = \{0\}$, hence $\sim \subseteq \ker \Phi^{\{0\}}$. The abstraction map also respects the colour assigned by the objective automaton.

▶ **Lemma 5** (Preservation of objective colouring). *If $h_J(\tau) = h_J(\tau')$, then $\lambda(\tau) = \lambda(\tau')$, for all histories τ, τ' and active coalitions J.*

Proof. The statement is immediate for $J = I$, since $\varphi_I(\tau) = \varphi_I(\tau')$ is a state q, so $h_I(\tau) = h_I(\tau') = (\{q\}, q)$ and $\lambda(\tau) = \lambda(\tau') = \lambda(q)$. Otherwise, if $J^{\Uparrow}$ is nonempty and $h_J(\tau) = h_J(\tau')$, then $\Phi^K(\tau) = \Phi^K(\tau')$ for all $K \in J^{\Uparrow}$, in particular $\Phi^I(\tau) = \Phi^I(\tau') = \{q\}$, and again $\lambda(\tau) = \lambda(\tau') = \lambda(q)$. ◀

Accordingly, we can map abstract states to colours via $\lambda^{\mathcal{H}}(\Phi, \varphi) := \lambda(\tau)$ for any history $\tau \in \Gamma^*$ such that $h(\tau) = (\Phi, \varphi)$.

An elementary, but important insight is that the abstract state of a history determines the set of abstract states of its information set.

▶ **Lemma 6** (Information commuting). *If $h_J(\tau) = h_J(\tau')$ then $h_J([\tau]_{\sim^J}) = h_J([\tau']_{\sim^J})$, for all histories τ, τ' and active coalitions J.*

Proof. Any histories τ, τ' with the same image under h_J share $\Phi^J(\tau) = \Phi^J(\tau') =: \Psi$, so their information sets map to:

$$\begin{aligned} h[\tau]_{\sim^J} &= \{(\Phi^J(\pi), \varphi) \mid \varphi \in \Phi^J(\tau)\} = \{(\Psi, \varphi) \mid \varphi \in \Psi\} \\ &= \{(\Phi^J(\tau'), \varphi) \mid \varphi \in \Phi^J(\tau')\} = h[\tau']_{\sim^J}. \end{aligned}$$ ◀

Regularity

To show that the abstraction map h can be constructed effectively, we prove that it is a homomorphism on the repeated game $\mathcal{G}$, as it preserves the move operations: for every pair of histories $\tau, \tau' \in \Gamma^*$, and every $c \in \Gamma$, if $h(\tau) = h(\tau')$, then $h(\tau c) = h(\tau' c)$. Towards this, we define operations on abstract states that mimic the information updates triggered either by local observations or by communication.

Intuitively, when a coalition J communicates with coalition L, it is as if J communicates with $J \cup L$. Therefore, we focus on communication between coalitions that are comparable with respect to inclusion. We define, for every increasing pair of active coalitions $J \subseteq L$, a lifting operator $\lceil\cdot\rceil_L^J \colon \mathrm{KSet}^L \to \mathrm{KSet}^J$ that maps every knowledge set Φ for L to a knowledge set $\lceil\Phi\rceil_L^J$ for J, by setting $\lceil\Phi\rceil_J^J := \Phi$ and

$$\lceil\Phi\rceil_L^J := \{\lceil\Phi, \varphi\rceil_L^J \mid \varphi \in \Phi\}, \quad \text{for all } L \in J^{\Uparrow}.$$

The construction uses an auxiliary operator that maps every abstract state $(\Phi^L, \varphi_L) \in \mathrm{KSet}^L \times \mathrm{Conf}_L$ for the larger coalition L to a configuration $\lceil\Phi, \varphi\rceil_L^J \in \mathrm{Conf}^J$ for the smaller J:

$$\lceil\Phi, \varphi\rceil_L^J := (\Psi^K)_{K \in J^{\Uparrow}} \quad \text{with } \Psi^K := \begin{cases} \lceil\Phi\rceil_L^K & \text{if } K \subseteq L; \\ \lceil\varphi^{K\cup L}\rceil_{K\cup L}^K & \text{otherwise.} \end{cases}$$

By unfolding the definition, we can see that the operators compose naturally.

▶ **Lemma 7** (Composition of lifting). *For all increasing coalitions $J \subseteq K \subseteq L$,*

$$\big\lceil\lceil\Phi\rceil_L^K\big\rceil_K^J = \lceil\Phi\rceil_L^J, \quad \textit{for every knowledge set } \Psi \in \mathrm{KSet}^J.$$

The following lemma states that the operator indeed captures that the information of a coalition L is revealed to smaller (hence, possibly less informed) coalition J.

▶ **Lemma 8** (Communication). *For coalition J and a move $c \in \Gamma$, consider the target coalition $L = \mathrm{sync}^J(c)$. Then, $\lceil\Phi^L(\tau c)\rceil_L^J = \Phi^J(\tau c)$, for every history $\tau \in \Gamma^*$.*

Proof. We proceed by induction starting with the grand coalition. As $\mathrm{sync}^I(d) = I$, the base case is trivial. For the induction step, suppose that the statement holds for all coalitions strictly larger than J. By unfolding the definitions, we obtain:

$$\begin{aligned} \lceil\Phi^L(\tau c)\rceil_L^J &= \big\{\big(\lceil\Phi^L(\tau c), \varphi\rceil_L^K\big)_{K \in J^{\Uparrow}} \mid \varphi \in \Phi^L(\tau c)\big\} && \text{(Set lifting, configuration)} \\ &= \big\{\big(\lceil\Phi^L(\tau' d), \varphi_L(\tau' d)\rceil_L^K\big)_{K \in J^{\Uparrow}} \mid \tau' d \sim^L \tau c\big\} && \text{(Lemma 3(i), Lemma 4)} \\ &= \big\{\big(\lceil\Phi^{L\cup K}(\tau' d)\rceil_{L\cup K}^K\big)_{K \in J^{\Uparrow}} \mid \tau' d \sim^L \tau c\big\}. && \text{(Abstract-state lifting)} \end{aligned}$$

Observe that for every $K \in J^{\Uparrow}$, we have $\mathrm{sync}^K(d) = \mathrm{sync}^{J\cup K}(d) = \mathrm{sync}^J(d) \cup \mathrm{sync}^K(d) = L \cup \mathrm{sync}^K(d)$. On the other hand $\mathrm{sync}^{L\cup K}(d) = \mathrm{sync}^L(d) \cup \mathrm{sync}^K(d) = L \cup \mathrm{sync}^K(d)$. Hence, the coalitions $L \cup K$ and K share the same target $T := \mathrm{sync}^{L\cup K}(d) = \mathrm{sync}^K(d)$, strictly larger than J. Therefore, we can develop the knowledge sets in the last expression as follows:

$$\begin{aligned}
\lceil \Phi^{L\cup K}(\tau' d)\rceil^K_{L\cup K} &= \lceil \lceil \Phi^T(\tau' d)\rceil^{L\cup K}_T\rceil^K_{L\cup K} && \text{(Induction hypothesis for } \mathrm{sync}^{K\cup L}(d)=T)\\
&= \lceil \Phi^T(\tau' d)\rceil^K_T && \text{(Lemma 7)}\\
&= \Phi^K(\tau' d) && \text{(Induction hypothesis for } \mathrm{sync}^K(d)=T).
\end{aligned}$$

In conclusion, $\lceil \Phi^L(\tau c)\rceil^J_L = \{\varphi_J(\tau' d) \mid \tau' d \sim^L \tau c\} = \{\varphi_J(\tau' d) \mid \tau' d \sim^J \tau c\} = \Phi^J(\tau c)$, according to Lemma 3(ii). ◀

With the lifting operators in place, we define update operations on configurations, knowledge sets, and ultimately on abstract states, for each active coalition.

For configurations, define $\mathrm{succ}_J \colon \mathrm{Conf}_J \times \Gamma \to \mathrm{Conf}_J$ as follows. For the grand coalition, this coincides with the automaton update: $\mathrm{succ}_I(\varphi, c) := \delta^{\mathcal{A}}(\varphi, c)$. For other coalitions J, the update is componentwise, based on synchronisation targets:

$$\mathrm{succ}_J(\varphi, c) := \big(\mathrm{Succ}^K(\varphi^{\mathrm{sync}^K(c)})\big)_{K\in J^\uparrow}.$$

Knowledge sets are updated by a partial function $\mathrm{Succ}^J \colon \mathrm{KSet} \times \Gamma \to \mathrm{KSet}^J$, defined on pairs (Φ, c) with $\Phi \in \mathrm{KSet}^L$ and $L = \mathrm{sync}^J(c)$:

- If J is autonomous ($J = L$), set

$$\mathrm{Succ}^J(\Phi, c) := \{\mathrm{succ}_J(\psi, d) \mid \psi \in \Phi,\ d^L = c^L\},$$

- Otherwise, for $L \neq J$, define

$$\mathrm{Succ}^J(\Phi, c) := \lceil \mathrm{Succ}^L(\Phi, c)\rceil^J_L.$$

The state update $\delta^{\mathcal{H}}_J \colon Q^{\mathcal{H}}_J \to Q^{\mathcal{H}}_J$ is defined as:

- If J is autonomous, then

$$\delta^{\mathcal{H}}_J((\Phi, \varphi), c) = (\mathrm{Succ}^J(\Phi, c),\ \mathrm{succ}_J(\varphi, c)),$$

- Otherwise, letting $L = \mathrm{sync}^J(c)$, set

$$\delta^{\mathcal{H}}_J((\Phi, \varphi), c) := (\mathrm{Succ}^J(\varphi^L, c),\ \mathrm{succ}_J(\varphi, c)).$$

▶ **Lemma 9** (Preservation of moves). *Let J be an active coalition. Then,*

$$h_J(\tau c) = \delta^{\mathcal{H}}_J(h(\tau), c), \quad \textit{for all } \tau \in \Gamma^*,\ c \in \Gamma.$$

Proof. We detail the inductive argument for knowledge sets to show that for $L = \mathrm{sync}^J(c)$,

$$\mathrm{Succ}^J(\Phi^L(\tau), c) = \Phi^J(\tau c).$$

The base case, with $I = \mathrm{sync}^I(c)$, concerns moves in the objective automata:

$$\begin{aligned}
\mathrm{Succ}^I(\Phi^I(\tau), c) &= \{\mathrm{succ}_I(\varphi, d) \mid \varphi \in \Phi^I(\tau), c^I = d^I\}\\
&= \{\delta(\delta(q_\varepsilon, \tau), c)\} = \{\delta(q_\varepsilon, \tau c)\} = \mathrm{Succ}^I(\tau c).
\end{aligned}$$

Inductive step: if J is autonomous,

$$\begin{aligned}
\mathrm{Succ}^J(\Phi^J(\tau), c) &= \{\mathrm{succ}_J(\varphi, d) \mid \varphi \in \Phi^J(\tau), d^J = c^J\} && \text{(Definition of Succ}^J\text{)}\\
&= \{\mathrm{succ}_J(\varphi_J(\tau'), d) \mid \tau' \sim^J \tau, d^J = c^J\} && \text{(Knowledge set } \Phi^J\text{)}\\
&= \{\big(\mathrm{Succ}^K(\Phi^{\mathrm{sync}^K(d)}(\tau'), d)\big)_{K\in J^\uparrow} \mid \tau' d \sim^J \tau c\} && \text{(Lemma 3(i))}\\
&= \{(\Phi^K(\tau' d))_{K\in J^\uparrow} \mid \tau' d \sim^J \tau c\} = \Phi^J(\tau c) && \text{(Induction hypothesis).}
\end{aligned}$$

Otherwise, if $J \neq L := \mathrm{sync}^J(c)$:

$$\begin{aligned}\mathrm{Succ}^J(\Phi^L(\tau), c) &= \lceil \mathrm{Succ}^L(\Phi^L(\tau), c) \rceil_L^J && \text{(Definition of } \mathrm{Succ}^J\text{)}\\ &= \lceil \Phi^L(\tau c) \rceil_L^J = \Phi^J(\tau c). && \text{(Induction hypothesis, Lemma 8)}\end{aligned}$$

The statement extends to configurations and abstract states by structural induction. ◀

With these ingredients in hand, let us define the automaton $\mathcal{H} = (Q^{\mathcal{H}}, \Gamma, Q^{\mathcal{H}}, \delta^{\mathcal{H}}, q_\varepsilon^{\mathcal{H}}, \delta^{\mathcal{H}})$ on the set $Q^{\mathcal{H}}$ of abstract states – also used as output alphabet – , with input alphabet Γ, transition function $\delta_0^{\mathcal{H}}$, initial state $q_\varepsilon^{\mathcal{H}} := h(\varepsilon)$ and an output function that returns the current state $\lambda^{\mathcal{H}} = \delta^{\mathcal{H}}$. This Mealy automaton $\mathcal{H}$ defines h. By projecting the output on its first component, we obtain, for every input history τ, the knowledge set $\Phi^{\{0\}}(\tau)$. In conclusion, the functions h and $\Phi^{\{0\}}$ are regular.

5.3 Knowledge Quotient

Now, we construct a new game from the automaton $\mathcal{H}$ by collapsing abstract states that share the same knowledge set. Concretely, we define the perfect-information game graph $\mathcal{K} = (Q^{\mathcal{K}}, (E_a^{\mathcal{K}})_{a\in A}, q_\varepsilon^{\mathcal{K}}, \lambda^{\mathcal{K}})$ with the following ingredients. The set of positions consists of knowledge sets: $Q^{\mathcal{K}} := \mathrm{KSet}^0 = \Phi^{\{0\}}(\Gamma^*)$. For every action $a \in A$, the set E_a contains the edge from Φ to Φ' if there exist abstract states (Φ, φ) and (Φ', φ') in $Q^{\mathcal{H}}$ and a move $c \in \Gamma$ such that $\delta^{\mathcal{H}}((\Phi, \varphi), c) = (\Phi', \varphi')$. The initial position is obtained from the initial abstract state $q_\varepsilon^{\mathcal{H}} = (\Phi_\varepsilon, \varphi_\varepsilon)$ by taking the first component $q_\varepsilon^{\mathcal{K}} := \Phi_\varepsilon$. As the objective colouring $\lambda^{\mathcal{A}}$ and the knowledge-set map $\Phi^{\{0\}}$ are both information consistent, we can lift the colouring from the configuration component of the states in $Q^{\mathcal{H}}$ defined in Lemma 5 to the knowledge set in their knowledge-set component: for any $\Phi \in Q^{\mathcal{K}}$, we assign $\lambda^{\mathcal{K}}(\Phi) := \lambda^{\mathcal{H}}(\Phi, \varphi)$ for some/any abstract state $(\Phi, \varphi) \in Q^{\mathcal{H}}$ with Φ in the first component.

The same game graph can be obtained directly from the original repeated game by taking the quotient $\Gamma/_{\ker \Phi}$. Indeed, $Q^{\mathcal{K}} = \Phi^0(\Gamma^*)$, $E_a = \{(\Phi^0(\tau), \Phi^0(\tau c)) \mid \tau \in \Gamma^*, c \in \Gamma\}$ for all $a \in A$, $q^\varepsilon = \Phi^0(\varepsilon)$, and $\lambda^{\mathcal{K}}(\Phi^0(\tau)) = \lambda(\tau)$, for all histories $\tau \in \Gamma^*$.

Let $k\colon \Gamma^* \to \mathrm{KSet}^0$ be the quotienting map that associates to every history its knowledge set: $k(\tau) := \Phi^{\{0\}}(\tau)$. Since k is information consistent with respect to $\sim$, it induces a map $\bar{k}\colon \Gamma^*/_\sim \to \mathrm{KSet}^0$ with $\bar{k}([\tau]) = k(\tau) = \Phi^0(\tau)$. The map $\bar{k}$ is actually a graph homomorphism from the information quotient $\mathcal{U}(\mathcal{G}) = \mathcal{G}^*/\sim$ of the original game to $\mathcal{K} = k(\mathcal{G})$, as it preserves edges, actions and the colouring λ.

As an immediate consequence of Lemma 6, the factor map describes a functional bisimulation.

▶ **Lemma 10** (p-morphism). *For every history $\tau c \in \Gamma^*$, and every $\tau' \in \Gamma^*$ with $k(\tau') = k(\tau)$, there exists a history $\tau'' \sim \tau'$ such that $k(\tau'' c) = k(\tau c)$.*

Accordingly, $\{([\tau], \Phi^0(\tau)) \in (\Gamma^*/\sim) \times \mathrm{KSet} \mid \tau \in \Gamma^*\}$ is a bisimulation between $\mathcal{U}(\mathcal{G})/\sim$ and $k(\mathcal{G})$ and, by Lemma 2, we can conclude that the games are equivalent.

▶ **Corollary 11** (Bisimulation). *The finite parity game $\mathcal{K}$ on $k(\Gamma^*)$ is bisimilar to the information quotient $\mathcal{U}(\mathcal{G})$ of the original FIP game on $\Gamma^*/\sim$. Moreover, for every winning strategy s in the parity game $\mathcal{K}$, the strategy $s \circ k = s \circ \Phi^{\{0\}}$ is winning in the FIP game $\mathcal{G}$.*

By positional determinacy of parity games, if there exists a winning strategy in $\mathcal{K}$, then there exists one represented by a labelling $s\colon Q^{\mathcal{K}} \to A$ of positions with actions. Recall that the positions of $\mathcal{K}$ correspond to knowledge sets $\Phi^0(\tau)$ output by the automaton $\mathcal{H}$ (with projection on the first component). By composing the output function of the automaton with the action labelling s, we thus obtain a winning strategy for the game $\mathcal{G}$ at the outset.

▶ **Theorem 12.** *The synthesis problem for FIP games with parity objectives is decidable. Whenever a winning strategy exists, we can effectively construct a Mealy automaton that defines one.*

Given a FIP game with n observers, the pre-processing step from Subsection 5.1 adds one observer, and thus the size $|\mathcal{K}|$ of the perfect-information game constructed in the reduction is $(n+1)$-fold exponential in the size of the FIP game. Note that the number of priorities for the parity objective does not change. Since perfect-information parity games can be solved in time at most exponential in the number of priorities (actually, in quasi-polynomial time in the number of nodes, see, e.g., [18, 20]), we derive an $(n+1)$-EXPTIME upper bound for the synthesis problem.

Theorem 12 extends to all visible objectives for which perfect-information games are decidable, such as mean-payoff or discounted-sum conditions.

6 Complexity Lower Bound

We show a matching lower bound for reachability objectives, by a reduction of the membership problem for alternating n-EXPSPACE Turing machines, which is $(n+1)$-EXPTIME-complete, to the synthesis problem for FIP games with n observers.

Intuitively, the FIP game simulates an execution of an alternating Turing machine M on an input word w of length ℓ, where the player chooses transitions in existential states, and Nature chooses transitions in universal states. The winning condition requires the player to announce successive configurations of M – consisting of the tape content (of n-fold exponential size), the head position, and the control state – until an accepting configuration is reached.

The main difficulty is to verify the consistency of these configurations using only polynomial-size Mealy automata. To address this, we use the structure of FIP games with imperfect information and n observers. Instead of checking transitions directly, the game allows Nature to challenge the equality of successive configurations. Nature may mark differing positions in the two configurations by sending observations to designated observers. These marks are hidden to the player until a communication occurs.

If the marked bits differ, the player may claim the marks refer to different positions, which reduces to checking inequality between two $(n-1)$-fold exponential-size numbers. The player does so by pointing out a differing bit in their binary encodings and announcing its address (a $(n-2)$-fold exponential-size number) – again recursively allowing Nature to challenge this claim. Each level of this recursive protocol uses one of the n observers. Eventually, the values are small enough to be verified by the objective automaton.

The encoding uses nested counters: a level-n counter is a sequence of bits, each followed by its address as a level-$(n-1)$ counter. This encoding is directly inspired by a similar definition in previous work [14, Section 4.2]. We define actions for the player to emit bits, counters, Turing machine transitions, and specific claims like r^{01} or r^{10} asserting bit differences. Nature's moves include sending observations (marks), initiating communication, and branching to verification components in the Mealy automaton.

Correct encoding is enforced by separate Mealy components that check syntactic properties: correct counter format, input encoding, control state updates, and that configurations follow transitions. Nature can branch to any component. The marking mechanism allows checking bit-equality claims, and address comparison proceeds recursively with up to n observers.

We argue that a small (polynomial-size) Mealy automaton can store the values of the marked bits in order to verify the player's claims (r^{01} or r^{10}). Storing two bits at each level (hence $2n$ bits) would suffice if Nature never challenges the format of the counters announced

by the player. However, if a challenge occurs, the game restarts at a lower level to check the counter format, which requires storing only $2(n-1)$ bits at that level – assuming no further challenges. Repeating this argument recursively, the total number of bits needed for storage is $O(n^2)$. Since n is fixed independently of the input word w and the Turing machine M, the resulting Mealy automata remain polynomial in size. This ensures that the overall reduction is polynomial-time.

We conclude that the player has a winning strategy in the FIP game if and only if the machine M accepts the input w.

▶ **Theorem 13.** *The synthesis problem for FIP games with n observers is $(n+1)$-EXPTIME-complete, both for parity and reachability objectives.*

7 Conclusion

We studied the problem of synthesising a finite-state process that must aggregate information from multiple sources, where the accessibility of a source is determined through interaction with the environment. This models dynamic networks with time-varying links and failures, and captures *convergecast* (single-sink data aggregation) tasks in which a designated process must fuse inputs from dispersed sensors while the environment controls when and with whom communication occurs. In game-theoretic terms, the question requires handling information trees with unbounded branching, beyond the reach of classical tree-automata approaches.

Our main contribution is a decision procedure for the case of one active player with visible objectives and n passive observers. Technically, we introduce a higher-order knowledge quotient that tracks nested coalitions and induces a p-morphism from the information quotient to a finite perfect-information parity game on knowledge sets; by positional determinacy, a winning strategy there yields a finite-state strategy for the original game.

The reduction is non-elementary in the number of observers, and we show tight complexity: with n observers, the synthesis problem is $(n+1)$-EXPTIME-complete. The upper bound holds for parity objectives and extends to other visible objectives (e.g., mean-payoff, discounted-sum), and the lower bound already for reachability.

Outlook

(i) Efficient implementability. Given a FIP strategy, can it be implemented by processes that exchange only bounded-length messages ? The answer is not always yes. For instance, consider one active player and two observers that all receive bit sequences; the player can communicate with the observers only upon input 1, and the specification requires the player to indicate whether the two observation histories are equal. Clearly, any protocol with a finite message space fails; each observer must reveal her full private history. If instead of equality, the specification asks whether the observation histories belong to a given regular relation, the task is solvable under FIP (the specification is indeed a winning strategy), but deciding implementability with finite messages corresponds to the *monadic decomposability of regular relations* [3]. Lifting such decomposition techniques to scenarios with intermediary communication events appears challenging, but possible.

(ii) Regular indistinguishability via two-tape automata. For indistinguishability relations recognisable by two-tape automata, information trees form a more general subclass of automatic trees than those arising from FIP. Do such information trees always admit a finite bisimulation quotient? A positive answer would yield decidability for one-player synthesis in games with imperfect information beyond full-information protocols.

References

1 André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical computer science*, 303(1):7–34, 2003. doi:10.1016/S0304-3975(02)00442-5.

2 Salman Azhar, Gary Peterson, and John Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *Journal of Computers and Mathematics with Applications*, 41:957–992, 2001. doi:10.1016/S0898-1221(00)00333-3.

3 Pablo Barceló, Chih-Duo Hong, Xuan-Bach Le, Anthony W. Lin, and Reino Niskanen. Monadic Decomposability of Regular Relations. 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 103:1–103:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.103.

4 D. Berwanger and L. Doyen. Observation and distinction. representing information in infinite games. *Theory of Computing Systems*, 67(1):4–27, 2023. doi:10.1007/s00224-021-10044-x.

5 D. Berwanger, A. B. Mathew, and M. van den Bogaard. Hierarchical information and the synthesis of distributed strategies. *Acta Informatica*, 55(8):669–701, 2018. doi:10.1007/s00236-017-0306-5.

6 Dietmar Berwanger, Lukasz Kaiser, and Bernd Puchala. A perfect-information construction for coordination in games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 387–398. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.387.

7 J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. doi:10.1090/S0002-9947-1969-0280205-0.

8 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM J. Comput.*, 51(2):17–152, 2022. doi:10.1137/17M1145288.

9 K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games of incomplete information. *Logical Methods in Computer Science*, 3(3:4), 2007. doi:10.2168/LMCS-3(3:4)2007.

10 A. Church. Logic, arithmetics, and automata. *Proc. Int. Congr. Math.*, pages 23–35, 1962.

11 Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: Crash failures. *Inf. Comput.*, 88(2):156–186, 1990. doi:10.1016/0890-5401(90)90014-9.

12 Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. *SIAM J. Comput.*, 34(5):1159–1175, 2005. doi:10.1137/S0097539703420675.

13 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 321–330. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.53.

14 B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Asynchronous games over tree architectures. In *Proc. of ICALP: Automata, Languages, and Programming*, LNCS 7966, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

15 Valentin Goranko and Martin Otto. Model theory of modal logic. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 249–329. North-Holland, 2007. doi:10.1016/S1570-2464(07)80008-5.

16 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of the fourteenth annual ACM symposium on theory of computing*, pages 60–65, 1982. doi:10.1145/800070.802177.

17 D. Harel and A. Pnueli. On the development of reactive systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

18 M. Jurdzinski, R. Morvan, and K. S. Thejaswini. Universal algorithms for parity games and nested fixpoints. In *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, LNCS 13660, pages 252–271. Springer, 2022. doi:10.1007/978-3-031-22337-2_12.

19 Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS '01*, pages 389–398. IEEE Computer Society Press, June 2001. doi:10.1109/LICS.2001.932514.

20 Karoliina Lehtinen, Pawel Parys, Sven Schewe, and Dominik Wojtczak. A recursive approach to solving parity games in quasipolynomial time. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/LMCS-18(1:8)2022.

21 Tal Mizrahi and Yoram Moses. Continuous consensus with ambiguous failures. *Theor. Comput. Sci.*, 411(34-36):3031–3041, 2010. doi:10.1016/J.TCS.2010.04.025.

22 M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980. doi:10.1145/322186.322188.

23 A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS: Foundations of Computer Science*, pages 746–757. IEEE, 1990.

24 M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

25 M. O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.

26 John H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984. doi:10.1016/0022-0000(84)90034-5.

27 D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011.

28 Sven Schewe. Distributed synthesis is simply undecidable. *Inf. Process. Lett.*, 114(4):203–207, April 2014. doi:10.1016/j.ipl.2013.11.012.

29 W. Thomas. On the synthesis of strategies in infinite games. In *Proc. of STACS: Symposium on Theoretical Aspects of Computer Science*, LNCS 900. Springer, 1995. doi:10.1007/3-540-59042-0_57.

30 W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

31 Thomas YC Woo and Simon S Lam. A lesson on authentication protocol design. *ACM SIGOPS Operating Systems Review*, 28(3):24–37, 1994. doi:10.1145/182110.182113.

Regulating Synchronous Data Exchange to Meet Control Flow and Data Specifications

Ashwin Bhaskar
Chennai Mathematical Institute, India

M. Praveen
Chennai Mathematical Institute, India
CNRS IRL ReLaX, Chennai, India

Abstract

When multiple software components interact via method calls, we may want to ensure that the order of invoked methods and the arguments provided adhere to some specification. The classic problem associated with interface automata checks for the existence of a mediator whose intention is to act as a buffer in between method invocations so that invocations do not go unanswered. We extend the base model underlying interface automata, enabling them to exchange integer values - one automaton generates an integer value and outputs it by firing a generating transition and another automaton receives the value by synchronously firing a receiving transition. Transitions in the automata can have guards with linear order constraints on the exchanged values, influencing which methods can or can not be invoked later. So the generated values influence the sequences of invocations that are enabled. We specify desirable properties of the sequence of method calls and the arguments passed to them using an extension of Linear Temporal Logic (LTL). We consider the interoperability problem, which is to check if it is possible to generate integer values in such a way that all enabled sequences satisfy the given specification.

We show that the interoperability problem is undecidable in general, even when there are only two participating automata. We show decidability in the case where guards on generating transitions can only have equality constraints on the exchanged value (but receiving transitions can continue to have linear order constraints). We model this problem as a game between two players, one trying to generate integer values such that violating sequences are disabled while the other player tries to dig out violating sequences that are enabled. Interoperability is equivalent to the first player having a winning strategy. We solve this game via a finite abstraction, which results in a symbolic game. We then show that winning strategies for the symbolic game can be translated to winning strategies for the original game over integers.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases Distributed Systems, Interface Automata, Registers, Parity Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.14

Funding *M. Praveen*: This author is partially supported by the Infosys foundation.

1 Introduction

There are many models for formalizing multiple software components interacting with each other. E.g. in [11], smart contracts are modeled using interface automata [5]. The classic question for interface automata is to check if there exists a mediator (that can buffer method calls among the participants) such that there are no unanswered method calls [5, 10]. In [1], smart contracts are modeled in the Behavior Interaction Priority (BIP) framework, which are finite state machines (FSMs) interacting via ports. The smart contracts in [1] are part of a supply chain for crude oil, and the goal is to model check properties such as "once the point-of-sale receives the oil batches from a vendor, the payment must be triggered". This is a typical temporal property saying one event must be followed by another.


© Ashwin Bhaskar and M. Praveen;
licensed under Creative Commons License CC-BY 4.0

FSMs in the BIP framework can exchange data in the form of arguments passed to method calls. We work with a model that also allows exchange of data (for example, an integer indicating the amount to be paid), but using synchronized transitions as in interface automata. In [1], control flow can be influenced by conditions such as "if the production at a vendor site is more than a specified limit, send an alert". We model these using guards on transitions, which comprise of linear order constraints on the data exchanged. The value that is exchanged thus influences which branches can be taken by the control flow. The interoperability problem is to check if the values can be generated in such a way that the control flow remains within a specified set. The set is specified by Constraint Linear Temporal Logic (CLTL), an extension of LTL that can check linear order constraints on data values. In the above supply chain example, this can check if there exists a value for the daily production limit such that vendor and point-of-sale work together ensuring that there is always enough inventory.

We benefit from game theoretic foundations of interface automata. To tackle the interoperability problem, we design a game between two players, SYSTEM and SPOILER. SYSTEM tries to choose values of the exchanged data (restricted to integers here) so that branches of control flow that can potentially violate the specification are disabled. SPOILER tries to pick out branches in the enabled part of control flow that can potentially violate the specification. If SYSTEM has a winning strategy, it provides integer values that will ensure that the specification is always met in all enabled branches of the control flow.

One difficulty that needs to be overcome is that branches in which the game can proceed depends on choices SYSTEM makes from an infinite domain (integers). We handle this by designing an abstraction that reduces the game to a finite one, while still maintaining enough information about existence of winning strategies. The high level idea of using abstraction to reduce infinite domains to finite ones is a known technique [7, 3, 2, 8]. But designing the abstraction must be done carefully so that peculiarities of the original model are correctly reflected in the resulting finite symbolic model. Our first contribution is coming up with a model that is expressive enough to specify interesting properties. An example is given in Section 4. Our second contribution is designing an abstraction [definition 19] such that the resulting finite symbolic model meets two goals. First, it is equivalent to the original model wrt the existence of winning strategies [Lemma 24]. Second, it can be solved by adapting known techniques [[3], Lemma 23].

Suppose the participants do not know each other's configurations and we need to check if they can still generate values based on the imperfect information they have. It turns out that if each participant sends integer values regularly, they can communicate their configuration information by encoding it in the integers they generate, thus communicating their configuration information to other participants and reducing it to the perfect information case. This trick will fail in other cases. It would be interesting to identify interesting fragments of the other cases where the interoperability problem makes sense.

Related work. In [10, 14], interface automata are extended with registers to handle data like we do here. But there, data values can be compared only for equality and its negation while we have linear order. Also, [10, 14] only consider specific safety properties; we allow properties expressible in CLTL. In [8], synthesis problem for specifications given as register automata over linearly ordered data domains (including $\mathbb{Z}$) are studied. They prove decidability for the existence of a winning strategy for one-sided games, where one of the players operates over a finite alphabet and the other manipulates data. The concrete game which we use to model the interoperability problem here is similar to such a one-sided game as even over

there, one of the players can only choose from a domain while the other can choose integer values. Also, we use techniques similar to theirs in order to deal with abstraction of the register values by tracking constraints between them and checking for the feasibility of such constraint sequences. However, synchronization between register automata is the central theme of our model and specification is based on a logic. In [8], register automata are used as specification mechanisms. Our techniques involve a combination of qualititative and quantitative specifications. Energy games [6, 4] provide a foundation for many such models.

2 Preliminaries

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ be the set of non-negative integers. We denote by $\lceil i \rceil_k$ the number i ceiled at k: $\lceil i \rceil_k = i$ if $i \leq k$ and $\lceil i \rceil_k = k$ otherwise. For integers n_1, n_2, we denote by $[n_1, n_2]$ the set $\{n \in \mathbb{Z} \mid n_1 \leq n \leq n_2\}$.

We recall here the definitions of constraint systems and Constraint LTL (CLTL) from [7] with some slight modifications. A constraint system $\mathcal{D}$ is of the form $(D, R_1, \ldots, R_n, \mathcal{I})$, where D is a non-empty set called the domain. Each R_i is a predicate symbol of arity a_i, with $\mathcal{I}(R_i) \subseteq D^{a_i}$ being its interpretation. For the sake of this paper, it suffices to describe CLTL formulas over constraint systems of the form $(D, <, =)$ where D is an infinite domain, $<$ is a total linear order over D and $=$ is the equality relation.

Let V be a finite set of variables, and A be a finite set of actions. A term is of the form $X^{-i}x$, where x is a variable and $i \geq 0$. A constraint c is of the form $t_i < t_j$ or $(t_i = t_j)$, where t_i, t_j are terms. The syntax of CLTL is given by the following grammar, where $a \in A$ and c is a constraint as defined above.

$$\phi ::= c \mid a \mid \neg\phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi$$

Let F_D denote the set of all mappings of the form $f : V \to D$. The semantics of CLTL is defined with respect to infinite sequences over $A \times F_D$ (also called concrete models in the following).

Given a concrete model $\alpha = (a_0, f_0)(a_1, f_1) \ldots \in (A \times F_D)^\omega$, the i^{th} position of α satisfies the atomic formula $a \in A$ (written as $\alpha, i \models a$) if $a = a_i$. Also, given $x, y \in V$ and $i_1, i_2 \in \mathbb{N}$, the i^{th} position of a concrete model α satisfies the constraint $X^{-i_1}x < X^{-i_2}y$ (resp. $X^{-i_1}x = X^{-i_2}y$) (written as $\sigma, i \models X^{-i_1}x < X^{-i_2}y$ (resp. $X^{-i_1}x = X^{-i_2}y$)) if and only if $i \geq i_1, i \geq i_2$ and $f_{i-i_1}(x) < f_{i-i_2}(y)$ (resp. $f_{i-i_1}(x) = f_{i-i_2}(y)$) hold. The semantics is extended to the rest of the syntax similar to the usual propositional LTL. We use the standard abbreviations $F\phi$ (resp. $G\phi$) to mean that ϕ is true at some position (resp. all positions) in the future. The X-length of a term $X^{-i}x$ is i. We say that a formula is of X-length k if it uses terms of X-length at most k. The formula $G(Fa \wedge X(X^{-1}x < y))$ will be true in the first position of a concrete model if, in all positions, the value of x is less than the value of y in the next position and if the action a holds infinitely often in the concrete model. In the rest of this paper when we refer to CLTL formulas, we shall restrict ourselves to CLTL formulas over the constraint system $(\mathbb{Z}, <, =)$ using just one variable d. The models for such formulas with just one variable can simply be thought of as sequences of the form $(a_0, v_0)(a_1, v_1) \ldots$ where $v_i \in \mathbb{Z}$.

3 Interface Automata with Data

We work with a model based on interface automata [5, 10]. We extend it so that integer values can be exchanged and stored.

▶ **Definition 1** (Finite State Register Machines (FSRMs)). *A finite state register machine* $\mathcal{M} = \langle B, q^0, R, T^E, T^H \rangle$ *where:*

- B *is a finite set of states,* $q^0 \in B$ *is the initial state and* R *is a finite set of registers,*
- $T^E \subseteq B \times \text{Test} \times \text{Asgn} \times B$ *is a set of external transitions, where* Asgn *is the power set of* R *and* Test *is the set of all formulas (guards) of the form* $g := r^i \leq r^j | r^i \leq d | g \wedge g | \neg g$ *where* $r^i, r^j \in R$ *and* d *is a formal parameter.*
- $T^H \subseteq B \times B$ *is the set of hidden transitions.*

We write $B_{\mathcal{P}}, R_{\mathcal{P}}, T^E_{\mathcal{P}}, T^H_{\mathcal{P}}$ to refer to the set of states, registers, external and hidden transitions of a particular FSRM $\mathcal{P}$. An R-valuation is a mapping from R to $\mathbb{Z}$. Given an R-valuation σ, an integer value v and a guard g, $\sigma, v \models g$ denotes that the formula g with each occurrence of $r \in R$ replaced by $\sigma(r)$ and each occurrence of d replaced by v, evaluates to true.

▶ **Definition 2** (Semantics of Finite State Register Machines). *The semantics of a finite state register machine* $\mathcal{M}$ *is a transition system defined as follows.*

- $C = B \times \mathbb{Z}^R$ *is the set of configurations,*
- $c^0 = (q^0, \{r \mapsto 0 \mid r \in R\})$ *is the initial configuration,*
- $(q, \sigma) \xrightarrow{(t,v)} (q', \sigma')$ *iff* $t = (q, g, u, q') \in T^E$ *such that* $\sigma, v \models g$ *and for each* $r \in R$, $\sigma'(r) = v$ *if* $r \in u$ *and* $\sigma'(r) = \sigma(r)$ *otherwise.*
- $(q, \sigma) \xrightarrow{t,*} (q', \sigma)$ *iff* $t = (q, q') \in T^H$.

In words, an external transition $t = (q, g, u, q')$ goes from configuration (q, σ) to (q', σ') storing the integer value v in registers belonging to u, provided t is enabled in σ: σ and the new integer value v should satisfy the guard g. Hidden transitions change states without changing register valuations. We refer to the pair (t, v) or $(t, *)$ as a *step* s.

▶ **Definition 3** (Traces of Finite State Register Machines). *Given a finite state register machine* $\mathcal{M}$*, we define a trace* τ *of* $\mathcal{M}$ *to be a finite or infinite sequence of alternating configurations and steps:* $c_0 s_0 \cdots c_{m-1} s_{m-1} c_m \cdots$ *where* $m \in \mathbb{N}$, $c_0 = c^0$, $c_i \in C$, $c_i \xrightarrow{s_i} c_{i+1}$ *and* s_i *is a step for all* $i \geq 0$*. We denote the set of all traces of* $\mathcal{M}$ *as* Traces.

A set of FSRMs can be composed. An output transition of one FSRM can synchronize with an input transition of another FSRM. To control which transitions can synchronize, we label them with a letter from a finite set of *actions*, along with one of the symbols $\{!, ?\}$. Transitions labeled with ! (resp. ?) are intended to be output (input) transitions. Two transitions can synchronize if they are labelled with the same action, one of them is additionally labeled with ! and the other one is additionally labeled with ?.

▶ **Definition 4** (Interface Register Transition System). *An interface register transition system* $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \text{ac} \rangle$ *where* $\overline{\mathcal{P}}$ *is a tuple of finite state register machines,* A *is a finite set of actions and* $\text{ac} : T^E_{\overline{\mathcal{P}}} \to \{!, ?\} \times A$ *is a labelling function. The notation* $T^E_{\overline{\mathcal{P}}}$ *stands for the union of the sets of external transitions of all the FSRMs in* $\overline{\mathcal{P}}$*.*

▶ **Definition 5** (Semantics of Interface Register Transition System). *The semantics of an interface register transition system* $\mathcal{S}$ *is a transition system defined as follows:*

1. *A configuration is a function* $c : \overline{\mathcal{P}} \to \cup_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times \mathbb{Z}^{R_{\mathcal{P}}}$ *such that* $c(\mathcal{P}) \in B_{\mathcal{P}} \times \mathbb{Z}^{R_{\mathcal{P}}}$ *for all* $\mathcal{P} \in \overline{\mathcal{P}}$*.*
2. $c^0 = \{\mathcal{P} \mapsto (q^0_{\mathcal{P}}, \{r \mapsto 0 \mid r \in R_{\mathcal{P}}\}) \mid \mathcal{P} \in \overline{\mathcal{P}}\}$ *is the initial configuration mapping all FSRMs to their initial states and all registers to* 0*.*

3. $c \xrightarrow{(a,t_o,t_i,v)}_{\mathcal{S}} c'$ *iff the following conditions are met:* $t_o \in T^E_{\mathcal{P}_o}$, $t_i \in T^E_{\mathcal{P}_i}$, $\mathcal{P}_o \neq \mathcal{P}_i$, $ac(t_o) = (!, a)$, $ac(t_i) = (?, a)$, $v \in \mathbb{Z}$, $(q_i, \sigma_i) \xrightarrow{t_i,v} (q'_i, \sigma'_i)$, $(q_o, \sigma_o) \xrightarrow{t_o,v} (q'_o, \sigma'_o)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P}_i)$ *from* (q_i, σ_i) *to* (q'_i, σ'_i) *and changing* $c(\mathcal{P}_o)$ *from* (q_o, σ_o) *to* (q'_o, σ'_o).
4. $c \xrightarrow{(a,t_o,*,v)}_{\mathcal{S}} c'$ *iff the following conditions are met:* $t_o \in T^E_{\mathcal{P}_o}$, $ac(t_o) = (!, a)$, $v \in \mathbb{Z}$, $(q_o, \sigma_o) \xrightarrow{t_o,v} (q'_o, \sigma'_o)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P}_o)$ *from* (q_o, σ_o) *to* (q'_o, σ'_o). *Additionally, for any transition* t_i *with* $ac(t_i) = (?, a)$, t_i *can not be fired in the configuration* c *with the value* v.
5. $c \xrightarrow{*,t,*,*} c'$ *iff the following conditions are met:* $t \in T^H_{\mathcal{P}}$, $(q, \sigma) \xrightarrow{t,*} (q', \sigma)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P})$ *from* (q, σ) *to* (q', σ).

In the transition mentioned in point 3 above, output transition t_o and input transition t_i from different FSRMs synchronize by exchanging the integer value v. In the transition of point 4, only an output transition from an FSRM fires and we think of the environment as receiving the value, provided none of the transitions in $\mathcal{S}$ can fire to receive the value v. In the transition in point 5, a hidden transition in FSRM $\mathcal{P}$ fires. We refer to tuples $(a, t_o, t_i, v), (a, t_o, *, v)$ and $(*, t, *, *)$ as steps.

▶ **Definition 6** (Traces of Interface Register Transition System). *Given an interface register transition system* $\mathcal{S}$*, we define a trace (or a finite trace)* τ *of* $\mathcal{S}$ *to be a finite alternating sequence of configurations and steps:* $c_0 s_0 \dots c_{m-1} s_{m-1} c_m$ *where* $c_0 = c^0$, $m \in \mathbb{N}$, $c_i \in C$, $c_i \xrightarrow{s_i}_{\mathcal{S}} c_{i+1}$ *and* s_i *is a step for all* $0 \leq i \leq m$*. We denote the set of all finite traces of* $\mathcal{S}$ *as* Traces*. An infinite trace* $\overline{\tau}$ *of* $\mathcal{S}$ *is similarly defined to be an infinite alternating sequence of configurations and steps. We denote the set of all infinite traces of* $\mathcal{S}$ *as* $\overline{\text{Traces}}$.

For a step (a, t_o, t_i, v) or $(a, t_o, *, v)$, its projection to actions and values is $\pi_{av}((a, t_o, t_i, v)) = \pi_{av}((a, t_o, *, v)) = (a, v)$. For a step $(*, t, *, *)$ with a hidden transition t, $\pi_{av}((*, t, *, *)) = \epsilon$, the empty sequence. We extend π_{av} to traces as $\pi_{av}(c_0 s_0 c_1 s_1 \cdots) = \pi_{av}(s_0)\pi_{av}(s_1) \cdots$. We say that a trace $\overline{\tau}$ satisfies a CLTL formula ϕ (denoted as $\overline{\tau} \models \phi$) if there are infinitely many external transitions fired along $\overline{\tau}$ and $\pi_{av}(\overline{\tau}) \models \phi$.

Two traces that differ only in hidden transitions can not be distinguished by CLTL formulas. We formalize this next.

▶ **Definition 7** (Hidden transitions invariance). *Suppose a finite trace* τ *is of the form* $\tau_1 \cdot c \cdot (*, t, *, *) \cdot \tau_2$*. Bypassing the step* $(*, t, *, *)$ *based on the hidden transition* t *will result in the sequence* $\tau_1 \cdot \tau_2$*. Let* τ^* *be obtained from* τ *by bypassing all hidden transitions occurring in* τ*. We say two traces* τ_1, τ_2 *are equivalent wrt hidden transitions (written as* $\tau_1 \sim_I \tau_2$*) if* $\tau_1^* = \tau_2^*$.

A step (a, t_o, t_i, v) can only be taken if the current configuration along with the new value v satisfies the guards in the transitions t_o, t_i. Choosing certain values can thus help in avoiding traces that violate a given CLTL specification. We denote by $T^O_{\mathcal{P}}$ the set of transitions $T^O_{\mathcal{P}} = \{t \in T^E_{\mathcal{P}} \mid ac(t) \in \{!\} \times A\}$.

▶ **Definition 8** (Regulator). *Given an interface register transition system* $\mathcal{S}$*, a regulator is a function* $\text{reg} : Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}} \to \mathbb{Z}$.

An infinite trace $\overline{\tau}$ conforms to a regulator reg if the following is true: for every prefix of the form $\tau(a, t_o, t_i, v)c'$ or $\tau(a, t_o, *, v)c'$, $v = \text{reg}([\tau]_{\sim_I}, t_o)$. We say that an interface register transition system $\mathcal{S}$ is interoperable satisfying a CLTL formula ϕ if there is a regulator reg such that all traces $\overline{\tau}$ that conform to reg satisfy ϕ.

In order to handle maximal finite traces, we add gadgets to the interface register transition system. For every action a, we add a sink state s_a. For an existing state q, suppose $g_1, \ldots, g_r$ are the guards on the input transitions from q labelled with a. We add an input transition from q to s_a labelled with the action a and the guard $\neg(g_1 \vee \ldots \vee g_r)$. This new transition is enabled if and only if none of the existing input transitions labelled by a are enabled. On s_a, we have a self-loop with an output transition labelled with a new action a'. On every existing state in all the FSRMs, we add self-loops with an input transition labelled with a'. This way, a maximal finite play is artificially extended to an infinite one by padding $(a')^\omega$. This can be easily identified in the CLTL specification and handled accordingly (like disallowing such actions a' as needed for the undecidability proof in appendix A).

In general, the interoperability problem for interface register transition system is undecidable. Given a 2-counter machine, we construct an interface register transition system with just two FSRMs such that the runs of the 2-counter machine are simulated by the traces of $\mathcal{S}$. It is an adaptation of similar undecidability proofs for games involving numbers, in which one player simulates a two counter machine and the other player catches mistakes if any during the simulation. For example, if the player simulating an incrementing transition increments by more than 1, it can be caught by the other player by checking that there is a value between the previous and incremented value. The details of the proof of undecidability can be found in appendix A.

4 Modeling Example

In this section, we provide an example of modeling an e-commerce system with the aim of checking whether a client and a server can be made to work with each other by regulating the interaction appropriately. The example is for most parts copied from a similar example given in [10]; we add a feature to it motivating the introduction of linear order constraints in the guards of the FSRMs, which are absent in [10]. For modeling this example, we assume Interface Register Transition Systems can exchange multiple integer values in one transition. Our results can be easily extended to handle this extension.

The example models a typical online store; one FSRM for the client and one for the server. Suppose the client starts by sending a *StartOrder* message containing its *id* and it expects to receive an id of a new order. A schematic of the FSRM modeling the client is shown below in Figure 1.

In the figure, an edge label such as $!StartOrder(ClientId)$ means that it is a transition labeled with the action *StartOrder* and the symbol ! (to indicate that it is a output action) and sends an integer value stored in the register named *ClientId*. The client then orders a number of items via the *AddToOrder* action, at the end of which it provides the payment information via the *PlaceOrder* action. Some edge labels in the figure are a quite long, so we have used shorthands such as $stmt_1$. These shorthands are expanded as below.

$$
\begin{aligned}
stmt_i &= AddToOrder(OrderId, ItemId_i, Qty_i) \\
stmt &= PlaceOrder(OrderId, CCNo) \\
rjct_i &= OutOfRange(ItemId_i, Min, Max); Qty_i < Min \vee Max < Qty_i \\
cnfm_i &= Confirmed(OrderId, Item_i, Qty_i)
\end{aligned}
$$

$$stmt^i_a = Process_Item(OrderId, ItemId)$$
$$stmt^i_b = SetQty(OrderId, ItemId, Qty_i)$$
$$stmt^i_c = ConfirmItem(OrderId, ItemId, Qty_i)$$
$$stmt^i_d = OutOfRange(OrderId, ItemId, Qty_i, Min, Max)$$
$$stmt_e = CloseOrder(OrderId, CCNo)$$

Then the client expects all items together with the quantities to be confirmed by the customer service. The client also accepts rebuttals from the customer service in case the number of items ordered is outside the minimum and maximum range allowed, via the *OutOfRange* message. This is a feature we have added here beyond what is described in [10], in order to demonstrate the utility of linear order constraints. The edge label $rjct_i$ above implements this feature by using a guard to test that the quantity is indeed out of range. In practice, e-commerce systems would allow the client to revise the order, but for the sake of simplicity here, we just move ahead by dropping those items that are out of range. It blocks in case that it does not receive the right confirmation for the remaining items. Then it announces that it is ready to close the order and expects to receive the result of the payment transaction.

The client service expects to receive the client's id, then it sends a confirmation and sends an id of a new order, together with a client verification. It is then prepared to repeat a loop in which it 1) receives an order of an item, 2) receives a quantity, 3) confirms the quantity or rejects as out of range and sends minimum and maximum quantity allowed. After that, it receives payment information, arranges payment via a third party service (invisible to the client and not modeled here), and sends the result of the payment transaction.

The client and customer service are not directly interoperable, since the client sends all items before accepting responses but the customer service sends responses for each item before accepting the next item. They can however still interoperate via a mediator that buffers the item orders and communicates with the two parties as expected. As done in [10], we restrict ourselves to the case where the number of items per order is bounded by a constant n, which becomes a parameter of the interoperability problem. The specification will say that every item is either acknowledged or out of range and that the order is closed and the result of the payment transaction is provided to the client. This can be easily specified in CLTL.

5 Modelling interoperability as a game

In the simulation of the two counter machine briefly mentioned at the end of Section 4 (details of which can be found in appendix A), we check that counter increments and decrements are done correctly using guards on output transitions, which ensure that the generated value is strictly between two previously generated values. It turns out that if we restrict guards on output transitions to prevent such checks, interoperability problem is decidable. Specifically, let guards on output transitions be those of the form $g := r^i \leq r^j | r^i = d | g \wedge g | \neg g$. From now on, we shall call this the restricted syntax. The new value being generated (represented symbolically by d in guards) can only be tested for equality ($r_i = d$, as opposed to $r_i \leq d$ in general) with register values. In other words, the restricted guards can check if the generated value is actually there in one of the registers, which is a natural test one may want to do. With guards on output transitions obeying this restricted syntax, the interoperability problem is decidable. As a first step towards proving decidability, we informally describe a game formulation of the interoperability problem.

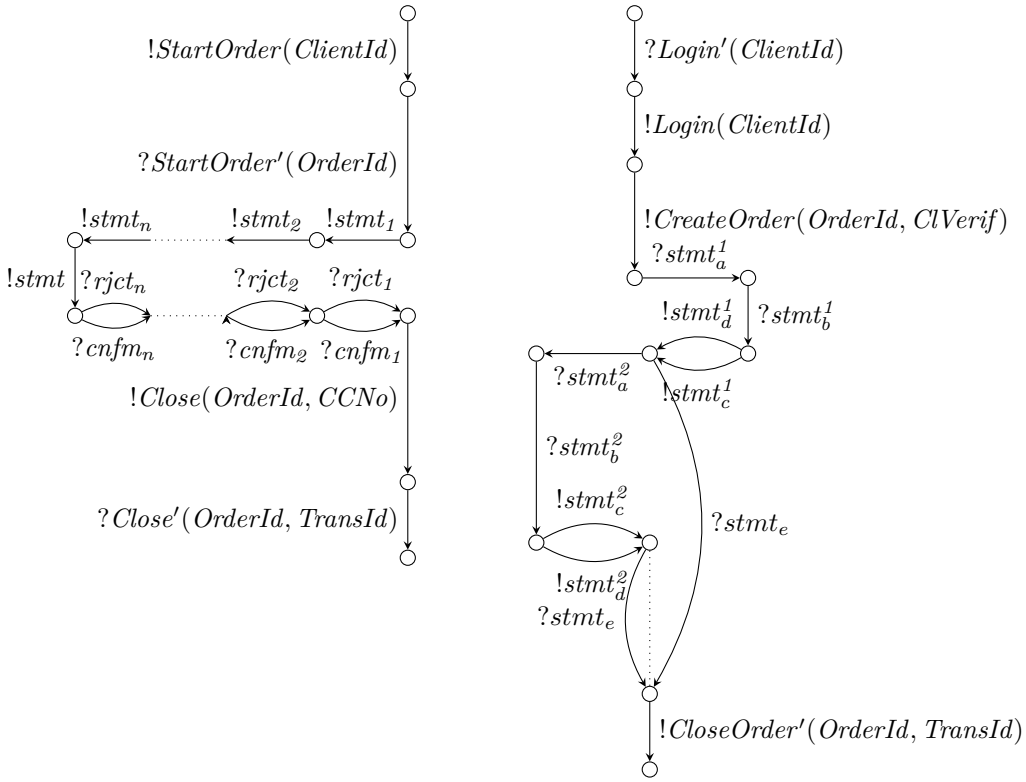


Figure 1 Modeling an e-commerce system. On the left is the client FSRM and on the right, customer service FSRM.

The game is played between SYSTEM and SPOILER, on the transition system which is the semantics of the given interface register transition system. SPOILER's goal is to pick out a trace $\overline{\tau}$ that violates the given CLTL specification. SYSTEM tries to prevent this by choosing values of the generated integers such that some transitions are disabled, thus restricting the choices available to SPOILER in the next move. The game begins at the initial configuration and SPOILER chooses an output transition t_o labeled with some action a enabled at the current configuration. SYSTEM chooses a value v based on the history of the play so far. SPOILER chooses a transition t_i enabled at some FSRM. The game moves to the updated configuration as per semantics. The game continues with the next round as before. SYSTEM wins the play if the infinite trace generated by the play satisfies the CLTL specification. The existence of a winning strategy for SYSTEM in this game is equivalent to the existence of a regulator.

To formally prove the above equivalence, we first formalize the game itself. A round ends with SPOILER choosing an input transition that is enabled at the current configuration. The next round begins wth SPOILER choosing an output transition that is enabled at the current configuration. We will combine these two moves of the spoiler and make it into a position of another game, which we call the *concrete game*. We denote by $T^I_{\overline{\mathcal{P}}}$ the set of transitions $T^I_{\overline{\mathcal{P}}} = \{t \in T^E_{\overline{\mathcal{P}}} \mid ac(t) \in \{?\} \times A\}$.

▶ **Definition 9** (Concrete game). *Given an interface register transition system $\mathcal{S}$, we associate with it a* concrete *game. The initial positions are of the form (c_0, t_o), where c_0 is any configuration that can be reached from the initial configuration c^0 with a finite sequence of hidden transitions and t_o is any output transition enabled at c_0. The set of other positions is $(T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times C \times T^O_{\overline{\mathcal{P}}}$. The set of* SYSTEM *moves is $\mathbb{Z}$.*

The concrete game starts at an initial position (c_0, t_o). From any position (c_0, t_o), (t_i, c, t_o) or $(*, c, t_o)$, SYSTEM makes a move by choosing a value $v \in \mathbb{Z}$. SPOILER can respond by moving to any position (t_i', c_2, t_o') (resp. $(*, c_2, t_o')$) provided the following conditions are satisfied:

- Output transition t_o and input transition t_i' are labeled with the same action, say a (resp. t_o is labeled with a and there is no input transition enabled at the configuration c to receive the value v),
- $c \xrightarrow{a,t_o,t_i',v}_{\mathcal{S}} c_1$ (resp. $c \xrightarrow{a,t_o,*,v}_{\mathcal{S}} c_1$), c_2 is reachable from c_1 by a sequence of hidden transitions and t_o' is an output transition enabled in c_2.

The game then continues as before from the new position. A play is an infinite sequence of alternating positions and moves: $p_0 v_0 p_1 v_1 p_2 \cdots$ such that successive positions satisfy the conditions above. Histories are finite prefixes of plays, ending at positions. With every play π, we can naturally associate an infinite sequence of actions and values $\pi_{av}(\pi)$. SYSTEM wins a play π if $\pi_{av}(\pi)$ satisfies the CLTL formula ϕ specified in the interoperability problem. Let H denote the set of all histories.

▶ **Definition 10** (Strategies). *A SYSTEM strategy is a function $f : H \to \mathbb{Z}$.*

A play $p_0 v_0 p_1 v_1 p_2 \cdots$ conforms to the strategy f if for all $i \geq 0$, $v_i = f(p_0 v_0 \cdots p_i)$. We say that f is a winning strategy for SYSTEM if all plays conforming to it are won by SYSTEM.

▶ **Proposition 11** (Interoperability and concrete game). *Let $\mathcal{S}$ be an interface register transition system such that guards on all the output transitions obey the restricted syntax and let ϕ be a CLTL formula. $\mathcal{S}$ is interoperable satisfying ϕ if and only if SYSTEM has a winning strategy in the corresponding concrete game.*

Proof. A regulator is a function $\text{reg} : Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}} \to \mathbb{Z}$. A strategy is a function $f : H \to \mathbb{Z}$. The result follows from the observation that there is a natural one-to-one correspondence between $Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}}$ and H. ◀

6 Symbolic interoperability game

In this section we prove that the interoperability problem for CLTL specifications is decidable when the guards on output transitions obey the restricted syntax.

We shall prove this by checking if SYSTEM has a winning strategy in the associated concrete game. This is in turn checked by designing a finite parity game that is equivalent to the concrete game with respect to SYSTEM having a winning strategy. We use positional determinacy of finite parity games to show this equivalence. In the rest of this section we elaborate on this.

In the concrete game, data exchanged between FSRMs come from $\mathbb{Z}$, an infinite set. We will have to simulate this in a game that can only use letters from a finite alphabet, for which we shall construct a symbolic abstraction.

We introduce the notion of partial frames and frames. We use frames (resp. partial frames) to capture the total pre-order (resp. not necessarily total pre-order) induced by the configurations and the data values along s consecutive positions in the concrete game. While a partial s-frame captures the pre-order induced by the configurations and data values appearing in the last s rounds of the concrete game upto the latest position of the SPOILER, an s-frame extends this to a total pre-order which includes the integer value generated by SYSTEM in its successive move as well.

The models of CLTL are infinite sequences over infinite alphabets. Frames, introduced in [7], abstract them to finite alphabets. Conceptually, frames and symbolic models as we will define here are almost the same as introduced in [7], where the authors used these notions to

solve the satisfiability problem for CLTL. Also, in [3], the authors use the notion of frames and partial frames in CLTL games to solve the single-sided realizability problem and the notions we use here are again, quite similar.

Given an interface register transition system $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \mathrm{ac} \rangle$ and a CLTL formula ϕ over A and a single variable d, we first define the notion of $\mathcal{S}$-variables and $\mathcal{S}$-terms.

The set of $\mathcal{S}$-variables $V_{\mathcal{S}} = \bigcup_{\mathcal{P} \in \overline{\mathcal{P}}} R_{\mathcal{P}} \cup \{d\}$. An $\mathcal{S}$-term is of the form $X^{-i}x$, where $x \in V_{\mathcal{S}}$ and $i \geq 0$. For $k \geq 0$, we denote by $T_{\mathcal{S}}[k]$ the set of all $\mathcal{S}$-terms of the form $X^{-i}x$, where $i \in [0, k]$.

▶ **Definition 12** (Frames, [7], [3, Definition 3]). *Given a number $s \geq 1$, an s-frame f is a pair $(a_f, \leq_f)$ where $a_f \in A$ and $\leq_f$ is a total pre-order [1] on the set of terms $T_{\mathcal{S}}[s-1]$.*

In the notation s-frame, s is intended to denote the size of the frame – the number of successive positions about which information is captured. The current position and the previous $s-1$ positions are considered, for which the terms in $T_{\mathcal{S}}[s-1]$ are needed. We denote by $<_f$ and $\equiv_f$ the strict order and equivalence relation induced by $\leq_f$: $x <_f y$ iff $x \leq_f y$ and $y \not\leq_f x$ and $x \equiv_f y$ iff $x \leq_f y$ and $y \leq_f x$.

We will deal with symbolic models that constitute sequences of frames. An s-frame will capture information about the first s positions of a model.

An s-frame meant for positions $i-s$ to $i-1$ may be followed by another s-frame meant for positions $i-s+1$ to i. The two frames must be consistent about the positions $i-s+1$ to $i-1$ that they share. The following definition formalizes this requirement.

▶ **Definition 13** (One-step compatibility, [7], [3, Definition 4]). *For $s \geq 1$, an s-frame f and an $(s+1)$-frame g, the pair (f, g) is one-step compatible if for all terms $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $t_1 \leq_g t_2$. For $s \geq 2$ and s-frames f, g, the pair (f, g) is one-step compatible if for all terms $t_1, t_2 \in T_{\mathcal{S}}[s-2] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $X^{-1}t_1 \leq_g X^{-1}t_2$*

Fix a number $k \geq 0$ and consider formulas of X-length k. A symbolic model is a sequence ρ of frames such that for all $i \geq 0$, $\rho(i)$ is an $\lceil i+1 \rceil_{k+1}$-frame and $(\rho(i), \rho(i+1))$ is one-step compatible. CLTL formulas can be interpreted on symbolic models, using symbolic semantics $\models_s$ as explained next. To check if the i^{th} position of ρ symbolically satisfies the atomic formula a, we check whether $\rho(i)$ is of the form $(a, \leq)$ for some pre-order $\leq$, and to check if the i^{th} position of ρ symbolically satisfies the atomic constraint $t_1 < t_2$ (where t_1, t_2 are terms), we check whether $t_1 < t_2$ holds according to the total pre-order $\leq_{\rho(i)}$ in the i^{th} frame $\rho(i)$. In formal notation, this is written as $\rho, i \models_s t_1 < t_2$ if $t_1 <_{\rho(i)} t_2$ holds. The symbolic satisfaction relation $\models_s$ is extended to all CLTL formulas of X-length k by induction on the structure of the formula, as done for propositional LTL. To check whether $\rho, i \models_s t_1 < t_2$ in this symbolic semantics, we only need to check $\rho(i)$, the i^{th} frame in ρ, unlike the CLTL semantics, where we may need to check other positions also. In this sense, the symbolic semantics lets us treat CLTL formulas as if they were formulas in propositional LTL and employ techniques that have been developed for propositional LTL. But to complete that task, we need a way to go back and forth between symbolic and concrete models.

Given a concrete model α, we associate with it a symbolic model $\mu(\alpha)$ as follows. Imagine we are looking at the concrete model through a narrow aperture that only allows us to view $k+1$ positions of the concrete model, and we can slide the aperture to view different portions. The i^{th} frame of $\mu(\alpha)$ will capture information about the portion of the concrete model visible when the right tip of the aperture is at position i of the concrete model (so the left tip

[1] a reflexive and transitive relation such that for all x, y, either $x \leq_f y$ or $y \leq_f x$

will be at $i - \lceil i \rceil_k$). Formally, the total pre-order of the i^{th} frame is the one induced by the valuations along the positions $i - \lceil i \rceil_k$ to i of the concrete model. The action occurring in the first component of each position of the symbolic model $\mu(\alpha)$ will be the same as the action appearing in the first component of the corresponding position of the concrete model α.

▶ **Lemma 14** ([7, Lemma 3.1], [3, Lemma 5]). *Let ϕ be a CLTL formula of X-length k. Let α be a concrete model and let $\rho = \mu(\alpha)$. Then $\alpha, 0 \models \phi$ iff $\rho, 0 \models_s \phi$.*

The symbolic model ρ induces an order $\leq_\rho$ on $\mathbb{N} \times V_{\mathcal{S}}$ as follows. Consider $i \in \mathbb{N}$ such that $\rho(i)$ contains an s-frame for some $s \geq 1$ and let $j \in \mathbb{N}$ be such that $j \leq i$ and $i - j \leq (s-1)$. We say $(i, x) \preceq_\rho (j, y)$ (resp. $(j, y) \preceq_\rho (i, x)$) if $x \leq_{\rho(i)} X^{j-i} y$ (resp. $X^{j-i} y \leq_{\rho(i)} x$). In other words, for the variables and positions captured in the frame $\rho(i)$, $\preceq_\rho$ is same as the total pre-order $\leq_{\rho(i)}$ over that frame. Now we define $\leq_\rho$ to be the transitive closure of $\preceq_\rho$ and $<_\rho$ to be the transitive closure of $\prec_\rho$. $\leq_\rho$ is now a partial order over $\mathbb{N} \times V_{\mathcal{S}}$.

For every concrete model, there is an associated symbolic model, but the converse is not true. We say that a symbolic model ρ admits a concrete model if there exists a concrete model α such that $\rho = \mu(\alpha)$. In particular, it is easy to see that if there are infinite chains of the form that we shall define below in a symbolic model then it does not admit a concrete model.

▶ **Definition 15** (Infinite chains, [7, Condition $C_{\mathbb{Z}}$]). *A symbolic model ρ is said to have infinite chains if there exist two infinite sequences $i_1 \leq i_2 \leq \cdots$ and $j_1 \leq j_2 \leq \cdots$ and two variables $x, y \in V_{\mathcal{S}}$ satisfying one of the following conditions:*

$$\begin{array}{ccccccccc} (i_1,x) & <_\rho & (i_2,x) & <_\rho & (i_3,x) <_\rho \cdots & & (i_1,x) & >_\rho & (i_2,x) & >_\rho & (i_3,x) >_\rho \cdots \\ \downarrow\wedge & & \downarrow\wedge & & \downarrow\wedge & \text{or} & \vee\downarrow^{\rho} & & \vee\downarrow^{\rho} & & \vee\downarrow^{\rho} \\ (j_1,y) & \geq_\rho & (j_2,y) & \geq_\rho & (j_3,y) \geq_\rho \cdots & & (j_1,y) & \leq_\rho & (j_2,y) & \leq_\rho & (j_3,y) \leq_\rho \cdots \end{array}$$

If a symbolic model ρ has an infinite chain as above, the integer values associated with $(i_2, x), (i_3, x) \ldots$ will all be distinct from each other and will be inside the interval from (i_1, x) to (j_1, y), which is impossible. Hence, absence of infinite chains is a necessary condition for a symbolic model to admit concrete models. We shall see later that this is also a sufficient condition for a subset of symbolic models

We next define partial frames and compatibility between a partial frame and a frame.

▶ **Definition 16** (Partial frames and compatibility). *For $s \geq 1$, a partial s-frame pf is a pair $(a_{pf}, \leq_{pf})$ where $a_{pf} \in A$ and $\leq_{pf}$ is a total pre-order on the set of terms $T_{\mathcal{S}}[s-1] \setminus \{d\}$. For $s \geq 0$, an s-frame f and a partial $(s+1)$-frame pf, the pair (f, pf) is one-step compatible if for all $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $t_1 \leq_{pf} t_2$. For $s \geq 2$, an s-frame $f = (a_f, \leq_f)$ and a partial s-frame $pf = (a_{pf}, \leq_{pf})$, the pair (f, pf) is one-step compatible if for all $t_1, t_2 \in T_{\mathcal{S}}[s-2] \setminus V_{\mathcal{S}}$, $X^{-1}t_1 \leq_{pf} X^{-1}t_2$ iff $t_1 \leq_f t_2$, and the pair (pf, f) is one-step compatible if $a_f = a_{pf}$ and for all $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus \{d\}, t_1 \leq_{pf} t_2$ iff $t_1 \leq_f t_2$.*

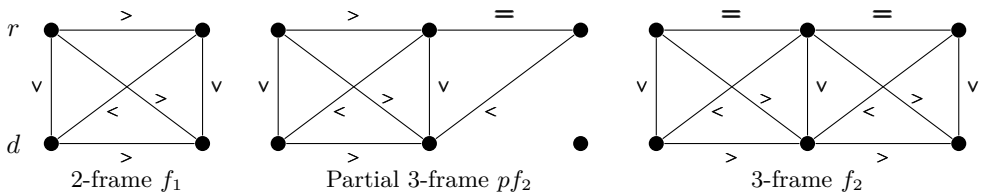


Figure 2 Frames, partial frames and one-step compatibility.

Figure 2 illustrates s-frames, partial s-frames and one-step compatibility graphically. Observe here that the pairs (f_1, pf_2) and (pf_2, f_2) are both one-step compatible. Note that the figure does not indicate some of the edges in the 3-frame and partial 3-frame which can be inferred using transitivity.

We now begin defining the symbolic game. We first define the notion of a symbolic configuration. A pair $(\overline{q_P}, f)$ where f is any $\lceil s \rceil_{k+1}$-frame for some $s \geq 1$, is called a symbolic configuration (denoted by sc). We call $(\overline{q_P}, pf)$ a partial symbolic configuration (denoted by psc) where pf is any partial $\lceil s \rceil_{k+1}$ -frame for some $s \geq 1$. Given a transition $t = (q, g, u, q')$, a tuple of states $\overline{q_P}$ and a $\lceil s \rceil_{k+1}$-frame f, we say t is enabled at $(\overline{q_P}, f)$ if the action label of t is same as the first component of f, $q \in \overline{q_P}$ and the frame f satisfies the guard g, or in other words, $r^i_{\mathcal{P}} \leq_f r^j_{\mathcal{P}}$ holds if $g = r^i_{\mathcal{P}} \leq r^j_{\mathcal{P}}$ and $r^i_{\mathcal{P}} \leq_f d$ holds if $g = r^i_{\mathcal{P}} \leq d$. We next define the notions of symbolic updates and symbolic steps which we shall use to design the rules of the finite parity game simulating the concrete game.

▶ **Definition 17** (Symbolic update). *Given an $\lceil s \rceil_{k+1}$-frame f, and an update $u_{\mathcal{P}}$ (or resp. a pair of updates $u_{\mathcal{P}_1}, u_{\mathcal{P}_2}$ with $\mathcal{P}_1 \neq \mathcal{P}_2$) we define the updated partial $\lceil s+1 \rceil_{k+1}$-frame pf to be such that (f, pf) is one-step compatible, $r_{\mathcal{P}} \equiv_{pf} X^{-1} r_{\mathcal{P}}$ for all $r_{\mathcal{P}} \notin u_{\mathcal{P}}$, $r'_{\mathcal{P}} \equiv_{pf} X^{-1} d$, for all $r'_{\mathcal{P}} \in u_{\mathcal{P}}$ and $r_{\mathcal{P}'} \equiv_{pf} X^{-1} r_{\mathcal{P}'}$ for all $r_{\mathcal{P}'} \in R_{\mathcal{P}'}$, for all $\mathcal{P}' \neq \mathcal{P}$ (resp. for a pair of updates, we update the register variables of both $\mathcal{P}_1$ and $\mathcal{P}_2$ and the remaining variables are equal to their previous values).*

▶ **Definition 18** (Symbolic step). *Given a symbolic configuration $sc = (\overline{q_P}, f)$, a partial symbolic configuration $psc' = (\overline{q_P}', pf')$, an action a and a pair of output and input transitions t_o, t'_i, where $\overline{q_P}, \overline{q_P}' \in \prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}}$, f is some $\lceil s \rceil_{k+1}$-frame and pf' is a partial $\lceil s+1 \rceil_{k+1}$-frame, we define a symbolic step $sc \xrightarrow{(a, t_o, t'_i)}_{\mathcal{S}} psc'$ iff the following conditions are met: $t_o = (q_{\mathcal{P}_o}, g_{\mathcal{P}_o}, u_{\mathcal{P}_o}, q'_{\mathcal{P}_o}) \in T^E_{\mathcal{P}_o}$, $t'_i = (q_{\mathcal{P}_i}, g_{\mathcal{P}_i}, u_{\mathcal{P}_i}, q'_{\mathcal{P}_i}) \in T^E_{\mathcal{P}_i}$, t_o, t'_i are enabled at sc, $\mathcal{P}_o \neq \mathcal{P}_i$, $ac(t_o) = (!, a)$, $ac(t'_i) = (?, a)$, $q'_{\mathcal{P}_o}, q'_{\mathcal{P}_i} \in \overline{q_P}'$ and pf' is obtained by updating f with the updates $u_{\mathcal{P}_o}$ and $u_{\mathcal{P}_i}$. We can define the symbolic step $sc \xrightarrow{(a, t_o, *)}_{\mathcal{S}} psc'$ in a similar manner.*

We know that any LTL formula ϕ can be converted to an equivalent non-deterministic Büchi automaton with an exponential number of states in the size of ϕ in EXPTIME [15]. Now, every non-deterministic Büchi automaton with n states can be converted to a deterministic parity automaton [9, Chapter 1] with number of states exponential in n and number of colours polynomial in n [13, Theorem 3.10]. Using these results, it is easy to see that given a CLTL formula ϕ, we can construct a deterministic parity automaton $\mathcal{A}_\phi$ with set of states Q and with number of colours l, accepting the set of all sequences of frames that symbolically satisfy ϕ, such that $|Q|$ is double exponential in the size of ϕ and l is exponential in the size of ϕ.

In order to construct a winning strategy for SYSTEM in the concrete game using a winning strategy for SYSTEM in the symbolic parity game that we shall construct, we also need to ensure that every sequence of frames resulting from a play of the game conforming to the winning strategy admits integer labellings for every variable in each of the frames. As done in [7] and [3], it is possible to construct a Büchi automaton of size polynomial in the size of the formula ϕ and hence a deterministic parity automaton $\mathcal{A}_{chain}$ of size exponential in the size of ϕ that checks for the absence of infinite chains in a symbolic model ρ satisfying ϕ. We shall use $\mathcal{A}_\phi$ and $\mathcal{A}_{chain}$ to define the parity winning condition of the symbolic game and we will see later that this ensures the existence of winning strategies with integer labellings for the sequence of frames along every play conforming to certain type of winning strategies.

▶ **Definition 19** (Symbolic game). *Given an interface register transition system $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \mathrm{ac} \rangle$ and a CLTL formula ϕ over A and a single variable d with X-length k, we associate with it a* symbolic *game. Let $\mathcal{F}$ denote the set of all s-frames for $s \in [0, k+1]$ and $\mathcal{PF}$ denote the set of all partial s-frames for $s \in [0, k+1]$. Let $\mathcal{A} = \mathcal{A}_\phi \times \mathcal{A}_{chain}$ be a deterministic parity automaton accepting the set of all sequences of frames that symbolically satisfy ϕ and do not contain infinite chains, with Q being the set of states, $q_I \in Q$ being the initial state and l being the number of colours. We define a parity game with initial positions $\{(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)\}$. Here, $\overline{q^0_{\mathcal{P}}}$ is the tuple of initial states of each of the FSRMs in $\mathcal{S}$, pf_0 is the partial 1-frame where all register variables are equal and t_o is some output transition whose source state is a current state in $\overline{q^0_{\mathcal{P}}}$. The set of other* SPOILER *positions is $\prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times (T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times \mathcal{PF} \times T^O_{\overline{\mathcal{P}}} \times Q$. The set of* SYSTEM *positions is $(\prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times (T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times \mathcal{F} \times T^O_{\overline{\mathcal{P}}} \times Q) \cup \{(\overline{q^0_{\mathcal{P}}}, f_0, t_o, q_I)\}$. Position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$ receives the same colour as that of the initial state q_I in $\mathcal{A}$ and positions $(t_i, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, $(*, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, $(t_i, \overline{q_{\mathcal{P}}}, f, t_o, q)$ and $(*, \overline{q_{\mathcal{P}}}, f, t_o, q)$ receive the same colour as that of state q in $\mathcal{A}$*

We denote the set of spoiler positions by P_{spo} and set of system positions by P_{sys} The symbolic game starts at an initial position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$. From any position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$, $(t_i, \overline{q_{\mathcal{P}}}, pf, t_o, q)$ or $(*, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, SYSTEM makes a move by choosing a position $(\overline{q^0_{\mathcal{P}}}, f_0, t_o, q_I)$, $(t_i, \overline{q_{\mathcal{P}}}, f, t_o, q)$ or $(*, \overline{q_{\mathcal{P}}}, f, t_o, q)$ where f_0 is the 1-frame with all register variables and the variable d equal to each other and f is a frame such that (pf, f) is one-step compatible and is such that f satisfies the guard g appearing in t_o.

SPOILER can respond by moving to any position $(t'_i, \overline{q_{\mathcal{P}}}', pf', t'_o, q')$ (resp. $(*, \overline{q_{\mathcal{P}}}', pf', t'_o, q')$) provided the following conditions are satisfied:

- There is a symbolic step from $(\overline{q_{\mathcal{P}}}, f)$ to $(\overline{q_{\mathcal{P}}}'', pf')$ on (a, t_o, t'_i) or $(*, t_o, t'_i)$ as defined in 18 and $\overline{q_{\mathcal{P}}}'$ is reachable from $\overline{q_{\mathcal{P}}}''$ by a sequence of hidden transitions. Formally, $(\overline{q_{\mathcal{P}}}, f) \xrightarrow{(a, t_o, t'_i)}_{\mathcal{S}} (\overline{q_{\mathcal{P}}}'', pf')$ (resp. $(\overline{q_{\mathcal{P}}}, f) \xrightarrow{(a, t_o, *)}_{\mathcal{S}} (\overline{q_{\mathcal{P}}}'', pf')$).
- Transition t'_o is enabled at the partial symbolic configuration $(\overline{q_{\mathcal{P}}}', pf')$
- There is a transition from q to q' on reading the frame f in the deterministic parity automaton $\mathcal{A}$.

The game then continues as before from the new position. A play is an infinite sequence of alternating SPOILER and SYSTEM positions: $p_0 p'_0 p_1 p'_1 p_2 p'_2 \cdots$ such that successive positions satisfy the conditions above. Histories are finite prefixes of plays, ending at SPOILER positions. SYSTEM wins a play of the game if the parity condition is satisfied by the play. Let H denote the set of all histories. Given a play π, let $\pi_f(\pi)$ be the projection of the play to the underlying sequence of frames contained in each SYSTEM position.

▶ **Definition 20** (Strategies). *A strategy for* SYSTEM *in the symbolic game is a function $st : H \to P_{sys}$.*

▶ **Definition 21** (Positional Strategies). *A positional strategy for* SYSTEM *in the symbolic game is a function $st : P_{spo} \to P_{sys}$.*

A play $p_0 p'_0 p_1 p'_1 p_2 p'_2 \cdots$ conforms to the strategy (resp. positional strategy) st if for all $i \geq 0$, $p'_i = st(p_0 v_0 \cdots p_i)$ (resp. $p'_i = st(p_i)$). We say that a strategy/positional strategy st is a winning strategy for SYSTEM if all plays conforming to it are won by SYSTEM. We abuse notation and say $t_1 \leq_{p'} t_2$ for $t_1, t_2 \in V_{\mathcal{S}}$ and $p' \in P_{sys}$ to mean $t_1 \leq_f t_2$ where f is the frame contained in position p'.

We associate a finitely branching infinite tree with a positional strategy st. Let Π_{st} be the collection of all plays that conform to strategy st and let H_{st} be the collection of all finite prefixes of plays in Π_{st} ending in a SPOILER position. Now let $\pi_{spo}(H_{st})$ denote the

projection of such histories to just the sequence of SPOILER positions. Now we define a tree associated with positional strategy st as T: $\pi_{spo}(H_{st}) \to P_{sys}$ given by $T(p_0 p_1 \dots p_i) = st(p_i)$. For every node η in T, its set of children corresponds to the set of SPOILER moves possible for the SYSTEM move $T(\eta)$. For a node η, the subtree T_η rooted at η is such that for all η', $T_\eta(\eta') = T(\eta \cdot \eta')$. A tree T is called *regular* if the set $\{T_\eta \mid \eta \in \pi_{spo}(H_{st})\}$ is finite, i.e., there are only finitely many subtrees up to isomorphism. Two nodes η, η' are said to be isomorphic if $T_\eta = T_{\eta'}$. It is easy to verify that the tree T associated with a positional winning strategy is a regular tree.

Given an infinite path $\pi \in \pi_{spo}(\Pi_{st})$ of the tree T, we associate with it, a symbolic model ρ, which is the infinite sequence of frames obtained from the labels of the nodes along that path.

We now give an automata-theoretic characterization for trees that are labellable. Using the fact that the trees associated with positional strategies are regular and using results from [3], we get that such trees are labellable if and only if the symbolic model associated with each infinite path of the tree does not have any infinite chain. Recall that the automaton $\mathcal{A}_{chain}$ we defined earlier, checks for absence of infinite chains in a symbolic model and therefore, we get the following lemma:

▶ **Lemma 22** ([3, Lemma 23]). *The tree T associated with a positional strategy st is labellable iff the symbolic model associated with every infinite path of T is recognized by the deterministic parity automaton $\mathcal{A}_{chain}$.*

Now, we know that if a positional strategy is winning, then the sequence of frames along every play conforming to the strategy must not have any infinite chains, hence no path in the tree associated with this strategy should have any infinite chains. Using Lemma 22, we then get that:

▶ **Lemma 23.** *The tree T associated with a positional winning strategy st is labellable.*

We now extend the map μ to go from histories h_c (resp. plays π_c) of the concrete game to histories h_s (resp. plays π_s) of the symbolic game. We define a function μ: $H_c \to H_s$ as follows: $\mu((c_0, t_0)v_0(t_0', c_1, t_1)v_1 \dots (t_i', c_{i+1}, t_{i+1}))$ equals $(\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I)(t_0', \overline{q_{\mathcal{P}}^1}, pf_1, t_1, q_1)(t_0', \overline{q_{\mathcal{P}}^1}, f_1, t_1, q_1) \dots (t_i', \overline{q_{\mathcal{P}}^{i+1}}, pf_{i+1}, t_{i+1}, q_{i+1})$ where pf_j is the partial $\lceil j+1 \rceil_{k+1}$-frame induced by the configurations $c_{j-\lceil j \rceil_{k+1}}, \dots c_j$ and the integer values $v_{j-\lceil j \rceil_{k+1}}, \dots v_{j-1}$ and f_j is the $\lceil j+1 \rceil_{k+1}$-frame induced by the configurations $c_{j-\lceil j \rceil_{k+1}}, \dots c_i$ and the integer values $v_{j-\lceil j \rceil_{k+1}}, \dots v_j$, $\overline{q_{\mathcal{P}}^j}$ is the tuple of states in configuration c_j and q_j is the state of DPA $\mathcal{A}$ on reading the sequence of frames $f_0 \dots f_{j-1}$ for all $0 \le j \le i$. μ is defined over plays of the concrete game in a similar manner. We now use all the above definitions and lemmas to establish the equivalence between the concrete and the symbolic games.

▶ **Lemma 24** (Equivalence between concrete and symbolic games). *SYSTEM has a winning strategy in the concrete game if and only if SYSTEM has a winning strategy in the symbolic game*

Proof. ($\Rightarrow$) Suppose SYSTEM has a winning strategy st in the CLTL game. We will construct a winning strategy st_p for SYSTEM in the symbolic game. Any history h_s of length 1 is of the form $(\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)$ and it is clear that $\mu((c_0, t_0)) = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I))$. Therefore, we define $st_p((\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)) =$ last position of $\mu((c_0, t_0) \cdot st((c_0, t_0)))$.

We assume inductively that the strategy st_p has been defined for all histories h_s in the symbolic game of length $j \le i$ and that there exists a history h_c of length j in the concrete game such that $\mu(h_c) = h_s$. Now, consider a history $h_s^{i+1} = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I) \dots$

$(t'_i, \overline{q_{\mathcal{P}}^{i+1}}, pf_{i+1}, t_{i+1}, q_{i+1})$ generated after $i+1$ rounds of the game. By induction hypothesis, there exists a history h_c^i of length i of the form $(c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i)$ such that $\mu(h_c^i) = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I)(t'_0, \overline{q_{\mathcal{P}}^1}, pf_1, t_1, q_1)(t'_0, \overline{q_{\mathcal{P}}^1}, f_1, t_1, q_1)\dots(t'_{i-1}, \overline{q_{\mathcal{P}}^i}, pf_i, t_i, q_i)$. Let $v_i = st((c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i))$ [The proof fails at this step if we allow the guards on the output transitions to use the full syntax instead of the restricted syntax. See note at the end of the proof]. Now, let $c_i \xrightarrow{a,t_i,t'_i,v_i}_{\mathcal{S}} c_{i+1}$. Thus, we now have $h_c^{i+1} = (c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i)(t'_i,c_{i+1},t_{i+1})$ such that $\mu(h_c^{i+1}) = h_s^{i+1}$ and therefore, $st_p(h_s^{i+1}) =$ last position of $\mu(h_c^{i+1}\cdot st(h_c^{i+1}))$.

Let π_s be any infinite play in the parity game that SYSTEM plays according to the strategy derived from st_p. By the inductive definition, there exists a play π_c in the concrete game played according to the strategy st such that $\pi_s = \mu(\pi_c)$. Now, since st is a winning strategy for the concrete game, π_c must be winning for SYSTEM in the concrete game, or in other words the concrete model underlying π_c must satisfy the CLTL formula ϕ. Therefore, by Lemma 14, the infinite sequence of frames along π_s must satisfy ϕ symbolically and since it admits an integer labelling it cannot have infinite chains. Therefore, π_s satisfies the parity winning condition of $\mathcal{A}$ or in other words, st_p is a winning strategy for SYSTEM in the symbolic game.

($\Leftarrow$) We know that if there is a winning strategy in a parity game then there must also be a positional winning strategy. Let st_p be such a positional winning strategy for SYSTEM in the parity (symbolic) game and let T_p be the tree associated with st_p. By Lemma 23, we know that T_p admits a labelling function L_p given by $L_p(\eta)\colon V_{\mathcal{S}} \to \mathbb{Z}$ for all $\eta \in \pi_{spo}(H_{st_p})$ such that $\hat{T}(\pi) = \mu(\hat{L}(\pi))$ for every infinite sequence $\pi \in \pi_{spo}(\Pi_{st_p})$. Now suppose $L_p((\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I))(x) = r$ for all $x \in V_{\mathcal{S}}$ (all $x \in V_{\mathcal{S}}$ must have the same label as they are equal wrt pf_0). We construct a new label function L'_p given by $L'_p(\eta)(x) = L_p(\eta)(x) - r$ for all $\eta \in \pi_{spo}(H_{st_p})$ and all $x \in V_{\mathcal{S}}$. Clearly, L'_p is also a valid labelling function satisfying $\hat{T}(\pi) = \mu(\hat{L'}_p(\pi))$ for every infinite sequence $\pi \in \pi_{spo}(\Pi_{st_p})$. We will use this to inductively construct a winning strategy st for the SYSTEM in the concrete game.

Consider a history h_c^i of length $i \geq 1$ in the concrete game. We define $st(h_c^i) = L'_p(\pi_{spo}(\mu(h_c^i)))(d)$. This labelling also respects the fact that the inital values of all the register variables are 0 in the concrete game and assigns values accordingly. Now consider any play π_c played according to strategy st. Clearly, by Lemma 14, the underlying concrete model $\alpha \models \phi$ as the symbolic model $\mu(\alpha)$ satisfies the formula ϕ symbolically as $\mu(\alpha)$ is the infinite frame sequence along the play $\pi_s = \mu(\pi_c)$ which is winning for the symbolic game and therefore satisfies the parity winning condition. Thus, $\pi_{av}(\pi_c) \models \phi$ or in other words, π_c is winning for SYSTEM and therefore, st is indeed a winning strategy for SYSTEM in the concrete game. ◀

As noted in the proof, the last output transition t_i may not actually be enabled at configuration c_i of the concrete game if the guard on t_i necessitates the newly generated value to be strictly between two consecutive integer values in c_i (notice that such a guard is used in the undecidability proof for simulating a counter increment, see Appendix A). The guard on t_i can indeed impose this condition if the guards on output transitions are allowed to use the full syntax instead of the restricted syntax.

We know that checking for the existence of a winning strategy of a player in a finite parity game is decidable. Thus, using Proposition 11, Lemma 24 and the decidability of parity games, we have the following main result:

▶ **Theorem 25** (Decidability of Interoperability). *Let S be an interface register transition system such that guards on all the output transitions obey the restricted syntax and let ϕ be a CLTL formula. The problem of checking whether S is interoperable satisfying ϕ is decidable.*

7 Discussion and Future Work

The deterministic parity automaton $\mathcal{A}$ that checks for bounded chains and symbolic satisfiability has size doubly exponential with respect to the size of the formula as already discussed before. The number of frames and partial frames is exponential in the number of registers of transition system $\mathcal{S}$. Therefore, the size n of the parity game graph is polynomial in the number of states and transitions in $\mathcal{S}$ and exponential in the number of registers. As we saw earlier, the number of colours l is exponential in the size of the formula. The interoperability problem for the decidable fragment can therefore be decided in $O(n^{log(l)})$. This gives us a 2EXPTIME upper bound for the interoperability problem.

We want to point out here that while it may seem that technically many of the ideas are similar those used in solving the realizability problem for CLTL in [3], the main difference is that we are addressing a problem about a transition system whereas in CLTL realizability, there is only a formula and no transition system. It is actually possible to encode the transition system as a CLTL formula. Using such an encoding, it is indeed possible to reduce the interoperability problem to the single-sided realizability problem for CLTL. But we know that single-sided realizability for CLTL over the integers is 2EXPTIME-complete. Hence, the upper bound we get will be doubly exponential in the size of the transition system. Based on the discussion above, the approach we have taken gives us a better upper bound in terms of the size of the transition system.

In this paper, we allow only integer values to be exchanged between finite state register machines. But one can also consider models where rational or real values could be exchanged. It turns out that the interoperability problem for interface register transition systems which exchange values coming from some dense, open domain is in fact decidable for the general syntax. For dense and open domains it turns out that satisfiability of any guard is guaranteed by its symbolic satisfiability ([7], [3]) (unlike in the case of integers, where it is not possible to satisfy a guard that demands that the newly generated value be between two registers whose values happen to be consecutive). Therefore, the equivalence established by Lemma 14 would hold even for the general syntax. Also, we do not need to check for the absence of unbounded chains for dense, open domains and therefore, we get decidability for the general syntax by reducing it to a parity game with in fact, a simpler parity winning condition.

We defined frames and partial frames to be total orders over the set of terms. But there could be pairs of variables that might never be compared. So, in principle, it is possible to try to avoid total orders and check if frames can be defined as partial orders. It would be interesting to see if this leads to better efficiency.

References

1 Sarra Alqahtani, Xinchi He, Rose Gamble, and Mauricio Papa. Formal verification of functional requirements for smart contract compositions in supply chain management systems. In *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020. doi:10.24251/HICSS.2020.650.

2 Ashwin Bhaskar and M Praveen. Constraint ltl with remote access. In *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2023)*, pages 41:1–41:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.FSTTCS.2023.41.

3 Ashwin Bhaskar and M Praveen. Realizability problem for constraint ltl. *Information and Computation*, 296:105126, 2024. doi:10.1016/J.IC.2023.105126.

4 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012. doi:10.1016/J.TCS.2012.07.038.

5 Luca De Alfaro and Thomas A Henzinger. Interface automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001. doi:10.1145/503209.503226.

6 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. doi:10.1007/978-3-642-15205-4_22.

7 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

8 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods in System Design*, 61(2):290–337, 2022. doi:10.1007/S10703-023-00435-W.

9 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

10 Lukáš Holík, Malte Isberner, and Bengt Jonsson. Mediator synthesis in a component algebra with data. In *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015, Proceedings*, pages 238–259. Springer, 2015. doi:10.1007/978-3-319-23506-6_16.

11 Gabor Madl, Luis Bathen, German Flores, and Divyesh Jadav. Formal verification of smart contracts using interface automata. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 556–563. IEEE, 2019.

12 Marvin L Minsky. Recursive unsolvability of post's problem of" tag" and other topics in theory of turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

13 Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3, 2007. doi:10.2168/LMCS-3(3:5)2007.

14 Ayleen Schinko, Walter Vogler, Johannes Gareis, N Tri Nguyen, and Gerald Lüttgen. Interface automata for shared memory. *Acta Informatica*, 59(5):521–556, 2022. doi:10.1007/S00236-021-00408-8.

15 Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

A Proof of undecidability stated in Section 3

An n-counter machine is a tuple $(B, q_{\text{init}}, n, \delta)$ where B is a finite set of states, $q_{\text{init}} \in B$ is an initial state, $c_1, \ldots, c_n$ are n counters and δ is a finite set of instructions of the form "$(q : c_i := c_i + 1; \text{goto } q')$" or "$(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$" where $i \in [1, n]$ and $q, q', q'' \in B$. A configuration of the machine is described by a tuple $(q, m_1, \ldots, m_n)$ where $q \in B$ and $m_i \in \mathbb{N}$ is the content of the counter c_i. The possible computation steps are defined as follows:

1. $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_i + 1, \ldots, m_n)$ if there is an instruction $(q : c_i := c_i + 1; \text{goto } q')$. This is called an incrementing transition.
2. $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_n)$ if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ and $m_i = 0$. This is called a zero testing transition.
3. $(q, m_1, \ldots, m_n) \to (q'', m_1, \ldots, m_i - 1, \ldots, m_n)$ if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ and $m_i > 0$. This is called a decrementing transition.

A counter machine is *deterministic* if for every state q, there is at most one instruction of the form $(q : c_i := c_i + 1; \text{goto } q')$ or $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ where $i \in [1, n]$ and $q', q'' \in B$. This ensures that for every configuration $(q, m_1, \ldots, m_n)$ there exists at most one configuration $(q', m'_1, \ldots, m'_n)$ so that $(q, m_1, \ldots, m_n) \to (q', m'_1, \ldots, m'_n)$.

For our undecidability results we will use deterministic 2-counter machines (i.e., $n = 2$), henceforward just "counter machines". Given a counter machine $(B, q_0, 2, \delta)$ and two of its states $q_{\text{init}}, q_{\text{fin}} \in B$, the reachability problem is to determine if there is a sequence of transitions of the 2-counter machine starting from the configuration $(q_{\text{init}}, 0, 0)$ and ending at the configuration $(q_{\text{fin}}, n_1, n_2)$ for some $n_1, n_2 \in \mathbb{N}$. It is known that the reachability problem for deterministic 2-counter machines is undecidable [12]. To simplify our undecidability results we further assume, without any loss of generality, that there exists an instruction $\hat{t} = (q_{\text{fin}} : c_1 := c_1 + 1; \text{goto } q_{\text{fin}}) \in \delta$.

Given a counter machine, we design an instance of the interoperability problem for an interface register transition system $\mathcal{S}$ consisting of just two FSRMs $\mathcal{P}$ and $\mathcal{Q}$ such that the counter machine reaches the halting configuration iff the answer to the interoperability problem for $\mathcal{S}$ is a "yes". Let T be the set of transitions of the counter machine. We construct an interface register transition system $\mathcal{S}$ whose set of actions $A = T \cup \{D\} \cup \{E\}$ where D, E are special actions. The FSRM $\mathcal{P}$ consists of states B of the counter machine and has three registers $r^0_{\mathcal{P}}, r^1_{\mathcal{P}}, r^2_{\mathcal{P}}$. The first register is intended to carry the value 0 and the last two are intended to carry the two counter values.

For every incrementing transition t of the counter machine from q to q', $\mathcal{P}$ has an output transition labeled with action t from q to q'. It has the guard $r^i_{\mathcal{P}} < d$ (syntactic sugar for $(r^i_{\mathcal{P}} \leq d) \ \wedge \ \neg(d \leq r^i_{\mathcal{P}})$), where c_i is the counter incremented by t. The value generated is stored the register $r^i_{\mathcal{P}}$. This ensures that the value generated by $\mathcal{P}$ while executing this transition is strictly greater than the current value of $r^i_{\mathcal{P}}$. But we need more work to ensure that the value generated by $\mathcal{P}$ is exactly one more than the current value of $r^i_{\mathcal{P}}$. We will describe later how to achieve this using the other FSRM $\mathcal{Q}$.

For every decrementing or zero testing transition t of the counter machine from q to q', $\mathcal{P}$ has an output transition labeled with action t from q to q'. It has the guard $d \leq r^i_{\mathcal{P}}$, where c_i is the counter decremented or tested for zero by t. The generated value is stored in the register $r^i_{\mathcal{P}}$.

The FSRM $\mathcal{Q}$ consists of states $B \cup (B \times T) \cup \{q_e\}$, (where B is the set of states of the counter machine), and four registers $r^0_{\mathcal{Q}}, r^1_{\mathcal{Q}}, r^2_{\mathcal{Q}}, r^3_{\mathcal{Q}}$. The first register is intended to store the value 0, the next two are intended to store the two counter values and the last one is intended to store the value generated by $\mathcal{P}$ to be used to detecting simulation errors.

For every incrementing transition t of the counter machine from q to q', $\mathcal{Q}$ has an input transition labeled with action t from q to q'. The received value is stored in the register $r^i_{\mathcal{Q}}$, where c_i is the counter incremented by t. There is also an input transition labeled t from q to (q', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q', t), there is an output transition labeled with a special action E (for "error") to the state q_e, with the guard $r^i_{\mathcal{Q}} < d \ \wedge \ d < r^3_{\mathcal{Q}}$. This forces $\mathcal{Q}$ to generate a value that is strictly between the old value of c_i and the new value $\mathcal{P}$ generated for that counter. If this transition is enabled, it means $\mathcal{P}$ generated a value that is more than one plus the old value of the counter c_i, which is an error. To let the FSRMs continue to operate infinitely, we add an output transition from (q', t) to itself labeled with a special action D (for "dummy") and an output transition from q_e to itself labeled E.

For every zero testing transition t of the counter machine from q to q', $\mathcal{Q}$ has an input transition labeled with action t from q to q'. The received value is stored in $r^i_{\mathcal{Q}}$, where c_i is the counter that is tested for zero by t. There is also an input transition labeled t from q to (q', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q', t), there is an output transition labeled with the action E with the guard $r^3_{\mathcal{Q}} \neq r^0_{\mathcal{Q}} \ \vee \ r^i_{\mathcal{Q}} \neq r^0_{\mathcal{Q}}$. This means that either the new counter value generated by $\mathcal{P}$ is non-zero or the old value of c_i was non-zero, both of which are errors. We add an output transition from (q', t) to itself labeled with the action D.

For every decrementing transition t of the counter machine from q to q'', $\mathcal{Q}$ has an input transition labeled with action t from q to q''. The received value is stored in $r^i_{\mathcal{Q}}$, where c_i is the counter that is decremented by t. There is also an input transition labeled t from q to (q'', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q'', t), there is an output transition labeled with the action E with the guard $(r^3_{\mathcal{Q}} < d \wedge d < r^i_{\mathcal{Q}})$. This forces $\mathcal{Q}$ to check that the value generated by $\mathcal{P}$ was less than the old value of c_i minus one, which is an error. We add an output transition from (q'', t) to itself labeled with the action D. At every state in $\mathcal{P}$, we add a transition to itself labeled with the action D and another transition to itself labeled with the action E. Now, adding gadgets to both these FSRMs as described at the end of Section 3 (to avoid finite traces) gives us a set of extra actions A'.

We now construct a CLTL formula ϕ (basically just an LTL formula over the set of actions $A \cup A'$ in this case) that excludes all strings that either containing E or the suffix $(a')^\omega$ for some $a' \in A'$. Among the remaining, it includes those which have at least one occurrence of D or one occurrence of a transition t that leads to the halting state q_{fin}.

▶ **Lemma 26.** *The halting state is reachable in the counter machine iff the interface register transition system constructed above is interoperable satisfying ϕ.*

Proof. If the halting state is reachable, we can construct a distributed regulator that updates the counter values exactly as the counter machine does. The only traces that respect this regulator are those which faithfully simulate the counter machine and those which execute dummy transitions. Both of these satisfy the formula ϕ.

If the interface register transition system is interoperable, consider a distributed regulator that makes it interoperable. If this regulator provides wrong values of the counter at any point during simulation, some trace respecting the regulator will execute some error transition, which does not satisfy ϕ, a contradiction. Hence, the regulator provides correct counter values. The traces that respect such a regulator are those that execute dummy transitions and the trace that faithfully simulates the counter machine. For this trace to satisfy ϕ, some transition leading to the halting state q_{fin} needs to be included, so the counter machine halts. ◀

Extending EFX Allocations to Further Multi-Graph Classes

Umang Bhaskar
Tata Institute of Fundamental Research, Mumbai, India

Yeshwant Pandit
Tata Institute of Fundamental Research, Mumbai, India

Abstract

The existence of EFX allocations is one of the most significant open questions in fair division. Recent work by Christodoulou, Fiat, Koutsoupias, and Sgouritsa ("Fair allocation in graphs," EC 2023) establishes the existence of EFX allocations for graphical valuations, when agents are vertices in a graph, items are edges, and each item has zero value for all agents other than those at its endpoints. Thus, in this setting, each good has non-zero value for at most two agents, and there is at most one good valued by any pair of agents. This marks one of the few cases when an exact and complete EFX allocation is known to exist for more than three agents.

In this work, we partially extend these results to multi-graphs, when each pair of vertices can have more than one edge between them. The existence of EFX allocations in multi-graphs is a natural open question given their existence in simple graphs. We show that EFX allocations exist, and can be computed in polynomial time, for agents with cancelable valuations in the following cases: (i) bipartite multi-graphs, (ii) multi-trees with monotone valuations, and (iii) multi-graphs with girth $(2t-1)$, where t is the chromatic number of the multi-graph. The existence of EFX in cycle multi-graphs follows from (i), (iii), and the known existence of EFX for three agents.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Fair Division, EFX, Multi-graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.15

Related Version *arXiv version*: `https://arxiv.org/abs/2412.06513`

Funding Both authors acknowledge the support of the Department of Atomic Energy, Government of India, under project no. RTI4001.

1 Introduction

In discrete fair division, we are given a set of indivisible items and a set of agents, with the goal of allocating the items *fairly* among the agents. Various formalisations of fairness have been studied in the literature, of which the most popular is *envy-freeness*, which requires that no agent prefers another's allocation to its own. However, envy-free allocations may not always exist, leading to extensive work on relaxations of envy-freeness. In this paper, we study the existence and computation of allocations that are envy-free up to any item, or EFX, in instances that can be represented as multi-graphs. The question of whether EFX allocations exist in general instances is possibly the most tantalizing open question in the fair allocation of resources, and many recent papers have focused on this question, leading to proofs of its existence in ever-larger classes of instances.

In our work, a fair division instance consists of a set of items that is to be allocated to a set of agents, with each item allocated to a single agent. Each agent has a monotone nondecreasing value function over the set of items. Given an allocation, agent i *envies* agent j if agent i's value for its own allocation is less than its value for the set of items allocated to j. An envy-free allocation is one where no agent envies another. Envy-free allocations do

© Umang Bhaskar and Yeshwant Pandit;
licensed under Creative Commons License CC-BY 4.0

not exist in general, and the closest relaxation then studied is envy-freeness up to any good, or EFX. An EFX allocation allows an agent to envy another, but the removal of *any* item from the envied agent's allocation should eliminate the envy as well.

EFX allocations were first studied by Caragiannis et al. [14], who established its relation to other notions of fairness, but left the question of existence open. If all agents have the same valuation, then an EFX allocation exists [25]. This can be extended to show the existence for two agents with distinct valuations. This was extended to multiple agents with two distinct additive valuations [24], then to three agents when at least one has an MMS-feasible valuation [3, 16], and recently to multiple agents with three distinct additive valuations [21]. A number of papers also study partial EFX allocations, when some items may remain unallocated, e.g., [17, 10], as well as approximately EFX allocations [5, 6, 15, 25].

Another widely studied relaxation of envy-freeness, weaker than EFX, is *envy-freeness up to one item* (EF1), which requires that any envy be resolved by removing *some* item from the envied agent's bundle [13, 23]. Unlike EFX, EF1 allocations are known to exist for both monotone and for some non-monotone valuations [23, 7, 12, 9, 11]. Other fairness notions have also been extensively studied in the literature [4].

Given the difficulties in establishing the existence of EFX allocations, researchers study special cases of structured valuations. EFX allocations are known to exist if agents have lexicographic preferences [20]. Another such class of structured valuations is *graphical valuations*, when the instance can be represented as a graph [18]. Here the agents correspond to the vertices and each item is an edge. The restriction is that each item has nonzero marginal value for only the agents at its end-points, and has zero value for all other agents. Thus if $e = \{i, j\}$, then for every other agent k, and any subset S of items, $v_k(S \cup \{e\}) = v_k(S)$. Further, the graph is simple, and thus there is at most one good valued by every pair of agents. For this case, when agents have monotone valuations over incident edges, Christodoulou et al. [18] show that EFX allocations exist, marking one of the rare cases where EFX allocations are shown to exist for more than three agents.

As mentioned in [18], graphical valuations model real-world scenarios with geographical constraints, where an item is only valued by nearby agents. For example, fishing rights in waters that are adjacent to two nations are such an item, valuable to the two nations but useless to others. Afshinmehr et al. [1] give the example of football matches in various formats; each match must be played at the home of one of the two teams in the match, and hence can be thought of as an item to be allocated to one of the two teams playing.

A natural question with graphical valuations is if an EFX *orientation* exists – an EFX allocation where goods are only allocated to agents at their endpoints. Christodoulou et al. [18] show that an EFX orientation may not exist, and it is NP-hard to determine if a given instance has an EFX orientation. Zeng and Mehta [28] characterize graphs that permit EFX orientations. Zhou et al. [29] study variants of EFX in the graphical setting, where an edge may be valued positively or negatively by its endpoints.

Given the existence of EFX in simple graphs, a natural question is if these results extend to multi-graphs when there are multiple edges between vertices (thus, pairs of agents can value multiple goods). Here a 2/3-approximate EFX allocation exists [5], beating the earlier bound of 0.618 for general additive valuations [6]. EFX allocations also exist in multi-graphs with *symmetrical* additive valuations, i.e., when agents have additive valuations, and both agents at the endpoints of a good have the same value for the good [22]. In independent and concurrent work, Afshinmehr et al. [1, 2] show that exact EFX allocations exist in bipartite multi-graphs and cycle multi-graphs. They also characterize when EFX orientations exist in bipartite graphs, and show it is NP-hard to decide if a given instance admits an EFX

orientation. Sgouritsa and Sotiriou [26] also independently present a number of results for multi-graphs when agents have general *monotone* valuations over their incident goods. We discuss these papers in Section 1.2.

1.1 Our Contribution

We study EFX allocations in multi-graphs and show that EFX allocations exist in large and important subclasses of multi-graphs. Our results are algorithmic, and for agents with cancelable valuations (defined in the next section), we show that an EFX allocation in these subclasses can be obtained in polynomial time. A feature of our algorithms is that they are iterative and fairly simple to state, based on the well-known cut-and-choose paradigm. While the algorithms are natural, the analysis is complicated and somewhat subtle.

1. Firstly, we show that for bipartite multi-graphs with cancelable valuations, an EFX allocation can be obtained in polynomial time.
2. Secondly, we show that for multi-trees, EFX allocations exist for *general monotone valuations*. Since multi-trees are also bipartite multi-graphs, for multi-trees with cancelable valuations, a polynomial time algorithm exists from the previous result.
3. Lastly, we generalize the result for bipartite multi-graphs, and show that for multi-graphs with girth at least $(2t-1)$, where t is the chromatic number of the graph, EFX allocations can be obtained in polynomial time for agents with cancelable valuations, given the colour classes.[1] Our algorithm here is a natural extension of the algorithm for bipartite multi-graphs, though the analysis is significantly more complicated.

Note that the chromatic number of a cycle multi-graph is 2 for even cycles and 3 for odd cycles. The existence of EFX allocations in even-length cycle multi-graphs follows from Result 1. The existence of EFX allocations in cycle multi-graphs of length 3 follows from prior work [16]. Result 3 implies the existence of EFX allocations in odd-length cycle multi-graphs of lengths at least 5. Hence, our work also shows the existence of EFX allocations for cycle multi-graphs of all lengths.

1.2 Comparison with Recent Work

Here, we compare our results with recent work on EFX existence in multi-graphs.

As noted earlier, Afshinmehr et al. [1, 2] independently and concurrently show the existence of EFX allocations in bipartite and cycle multi-graphs under additive valuations. Similar to earlier work [18], their algorithm for bipartite multi-graphs operates in two phases: the first constructs an EFX orientation with certain properties, and the second transforms it into a complete allocation using "safe" vertices. While their approach and analysis are somewhat involved, we present a simpler and efficient algorithm with a more straightforward analysis for *cancelable* valuations, which generalize additive valuations. Moreover, our algorithm for bipartite multi-graphs naturally extends to show the existence of EFX allocations in multi-graphs with chromatic number t and girth at least $2t-1$. This extension avoids the challenging case – highlighted in Section 5 of [1, 2] – where there are two envied vertices with the multi-edge between them unallocated, which poses a key difficulty in extending results to general multi-graphs. In contrast, the algorithm, as stated in Afshinmehr et al., does not immediately lend itself to any such natural extension beyond bipartite multi-graphs. Afshinmehr et al. also establish the existence of EFX in cycle multigraphs, which follows as a special case of our result for t-chromatic multigraphs with girth at least $2t-1$.

[1] Note that approximating the chromatic number of a graph within $n^{1-\epsilon}$ for all $\epsilon > 0$ is NP-hard [30].

Another independent and concurrent work by Sgouritsa and Sotiriou [26] proves EFX existence for certain multi-graph classes under *monotone* valuations. They show that EFX allocations exist for: (i) multi-graphs where each vertex has at most $\lceil \frac{n}{4} \rceil - 1$ neighbours (where n is the total number of vertices), and (ii) multi-graphs with girth at least 6. In follow-up work on Arxiv, they extend these results to bipartite multi-graphs [27]. These results extend many of ours to monotone valuations, except for multi-graphs with chromatic number 3 and girth 5 (i.e., $t = 3$), where we establish EFX existence. As in other papers [18, 1, 2], the algorithms by Sgouritsa and Sotiriou follow a multi-phase approach: the first phase computes an initial allocation (orientation) satisfying certain properties, the second aims to reduce the number of envied vertices, and the third assigns the remaining unallocated bundles to non-envied (safe) vertices who value them at zero. Our algorithms offer a different and simpler approach to those proposed in these other papers.

2 Notation and Preliminaries

Multi-graphs. A multi-graph $G = (V, E)$ is a graph with possibly multiple edges between each pair of vertices. A function r maps each edge $e \in E$ to an unordered pair of vertices, which are its endpoints. We use $E_{u,w}$ for the set of parallel edges between the vertices u, w (hence $E_{u,w} = E_{w,u}$). We define $n := |V|$ and $m := |E|$. For a vertex u, N_u is the set of neighbours $\{w \in V : \exists e \in E,\, r(e) = \{u, w\}\}$.

Given a multi-graph, we can define paths and cycles as in simple graphs without parallel edges. A path $P = (v_1, \ldots, v_k)$ is a sequence of distinct vertices so that each pair of consecutive vertices are neighbours. The length of a path is one less than the number of vertices. A cycle $C = (v_1, \ldots, v_{k-1}, v_k)$ is a sequence of distinct vertices so that each pair of consecutive vertices are neighbours, and v_1, v_k are neighbours.

Fair division. In a fair division instance on a multi-graph $G = (V, E)$, each vertex $u \in V$ corresponds to an agent u, and each edge e corresponds to a good. Throughout this article, we will use the terms agents and vertices interchangeably, and the terms items and edges interchangeably. Each agent u has a valuation function $v_u : 2^E \to \mathbb{Z}_+$ that maps subsets of goods to an integer value. A *bundle* $S \subseteq E$ is simply a subset of goods.

The multi-graph restricts the valuation functions in the following way: Each agent $u \in V$ has zero marginal value for any good not adjacent to it. Thus for a good g if $u \notin r(g)$, then for any subset of goods $S \subseteq E$, $v_u(S \cup \{g\}) = v(S)$. We say that agent u *values* a good g if $u \in r(g)$, i.e., the edge g is adjacent to u, else it does not value the good g. Extending this definition, we say that agent u values a bundle $S \subseteq E$ if S contains some good that u values, else u does not value the bundle S.

An allocation $A = (A_u)_{u \in V}$ is a partition of the set of items (or edges) among the agents. A partial allocation is a partition of a subset of items. An allocation A is *envy-free* (EF) if for all agents u, w, $v_u(A_u) \geq v_u(A_w)$. An allocation A is *envy-free up to any good* (EFX) if for all agents u, w, either $v_u(A_u) \geq v_u(A_w)$, or $v_u(A_u) \geq v_u(A_w \setminus \{x\})$, for all $x \in A_w$. An orientation is an allocation where each item is allocated to an agent that values it (thus if $r(e) = \{u, w\}$, then $e \in A_u \cup A_w$).

Given an allocation A (or a partial allocation), we can define an *envy graph* G_A among the agents: The vertices of G_A are the agents, and there is a directed edge (u, w) if agent u envies agent w, i.e., $v_u(A_u) < v_u(A_w)$. Note that an envy graph is simple.

Given an allocation A and the corresponding envy graph G_A, let $C = (u_1, u_2, \ldots, u_k)$ be a directed cycle in the envy graph. We use the term *resolving the envy cycle* C to refer to the modification that exchanges bundles along the cycle: agent u_k gets the bundle currently allocated to u_1, and each agent u_i gets the bundle currently allocated to u_{i+1}, for $i < k$. The modified allocation is called A^C.

Valuation functions. We will be interested in a few different classes of valuation functions. A valuation function $v : 2^E \to \mathbb{Z}_+$ is *monotone nondecreasing* (or just monotone) if for every $S \subseteq E$ and $g \in E$, $v(S \cup \{g\}) \geq v(S)$.

A monotone valuation function is *cancelable* if for $S, T \subseteq E$ and $g \notin S \cup T$, if $v(S) \geq v(T)$ then $v(S \cup \{g\}) \geq v(T \cup \{g\})$. These functions were defined in prior work on EFX allocations for four agents [10]. We only consider monotone cancelable functions, and hence say cancelable to mean monotone cancelable functions. We in particular need the following two properties of cancelable functions.

▶ **Proposition 1.** *Let v be a cancelable function. Then given disjoint sets of goods S_1, S_2, T_1, T_2, and $v(S_1) \geq v(T_1)$, $v(S_2) \geq v(T_2)$, it follows that $v(S_1 \cup S_2) \geq v(T_1 \cup T_2)$.*

Proof. By cancelability, $v(S_1 \cup S_2) \geq v(T_1 \cup S_2)$ and $v(T_1 \cup S_2) \geq v(T_1 \cup T_2)$, and hence $v(S_1 \cup S_2) \geq v(T_1 \cup T_2)$. ◀

For two agents with monotone valuations, EFX allocations are known to exist [25]. However, even for two agents with submodular valuations, computing an EFX allocation is PLS-complete and requires an exponential number of queries [19]. For cancelable valuations, we can compute an EFX allocation in polynomial time [8, 19].[2]

▶ **Proposition 2.** *For two identical agents with cancelable valuations, an EFX allocation can be computed in polynomial time.*

Cut and Choose. Cut and choose is a standard procedure to obtain an EFX allocation for two agents. Given agents u and w and a set of items S, one agent "cuts" and the other agent "chooses". If agent u "cuts", we assume there are two identical agents with valuation function v_u, and use the algorithm from Proposition 2 to obtain a partition (S^1, S^2) of S that is EFX for identical agents with valuation function v_u. Then agent w "chooses" the bundle with a higher value for it, while agent u is assigned the other bundle. The allocation is clearly envy-free for agent w, and is EFX for agent u. Note that for two agents with cancelable valuations, this gives us a polynomial time algorithm to obtain an EFX allocation.

The cut-and-choose procedure is central to our algorithm. In Section 3, when we deal with bipartite multi-graphs $G = (L \cup R, E)$, for allocating a set of parallel edges $E_{u,w}$ with $u \in L$, $w \in R$, agent $w \in R$ will always cut (thus the agent in the right bipartition always cuts). In Section 5, when we deal with t-chromatic multi-graphs, and partition the vertex set $V = C_1 \cup \ldots \cup C_t$ where vertices in C_i are the same colour, we think of the vertices in C_i as being to the left of C_{i+1}. For a set of parallel edges $E_{u,w}$, as in the bipartite case, the vertex to the *right* will always cut.

We use $\textsc{Cut}(u, S)$ to refer to the use the algorithm from Proposition 2 to obtain a partition (S^1, S^2) of S that is EFX for identical agents with valuation function v_u.

[2] The actual result holds for the more general class of *weakly well-layered* valuations.

3 EFX Allocations in Bipartite Multi-graphs

We first show that for bipartite multi-graphs and agents with cancelable valuations, an EFX allocation always exists, and can be computed in polynomial time. Let $V = L \cup R$ be the bipartition of the vertex set. We first define some notation we will use for this case.

Structures

For each vertex $u \in L$, we define the *structure* St_u as the subgraph induced by $u \cup N_u$. Note that since $N_u \subseteq R$, all edges in St_u are between u and vertices in N_u. We say u is the *root* of the structure. Each iteration in our algorithm will *resolve* the structure St_u, for some $u \in L$, by which we mean that it will assign all the edges in St_u. We also say that a vertex u is resolved to mean that St_u is resolved.

For allocating a set of parallel edges $E_{u,w}$ with $u \in L$, $w \in R$, agent $w \in R$ will always cut (thus the agent in the right bipartition always cuts). For each $w \in N_u$, let $(E^1_{u,w}, E^2_{u,w}) = \textsc{Cut}(w, E_{u,w})$ be the partition of $E_{u,w}$ returned when agent w cuts. Define $S_{u,w} := \arg\max\{v_u(E^1_{u,w}), v_u(E^2_{u,w})\}$ as the bundle preferred by $u \in L$, and $T_{u,w} := \arg\max\{v_w(E^1_{u,w}), v_w(E^2_{u,w})\}$ as the bundle preferred by w. Ties are broken arbitrarily to ensure that $S_{u,w}$ and $T_{u,w}$ are single bundles. If either agent is indifferent between the two bundles, we break ties so that $S_{u,w} \neq T_{u,w}$. Further, we define $\bar{S}_{u,w} = E_{u,w} \setminus S_{u,w}$ and $\bar{T}_{u,w} = E_{u,w} \setminus T_{u,w}$.

For a structure St_u, we further define a *favourite* neighbour f_u as follows:

$$f_u := \arg\max_{w \in N_u} v_u(S_{u,w})\,.$$

We break ties arbitrarily to ensure that f_u is a single vertex. Then f_u is a neighbour who offers u the highest-value bundle after cutting the adjacent edges. The other neighbours are simply called *ordinary* neighbours.

Our algorithm is then simple to describe, and is formally given as Algorithm 1. We consider the vertices in L in turn. For vertex $u \in L$, we consider the structure St_u and resolve it, as follows. Each neighbour $w \in N_u$ cuts the bundle $E_{u,w}$. Each *ordinary* neighbour $w \neq f_u$ gets their higher-value bundle $T_{u,w}$. Let $\mathrm{LO}_u = \bigcup_{w \in N_u \setminus \{f_u\}} \bar{T}_{u,w}$ be the union of the left-over goods from the ordinary neighbours of u. For the favourite neighbour f_u, if $S_{u,f_u} \neq T_{u,f_u}$ – both u and f_u prefer different bundles when f_u cuts E_{u,f_u} – then u gets $S_{u,f_u} \cup \mathrm{LO}_u$, and f_u gets T_{u,f_u}. In this case, everyone gets their largest value bundle, and there is no envy. On the other hand, if $S_{u,f_u} = T_{u,f_u}$ – both u and f_u prefer the same bundle when f_u cuts E_{u,f_u} – then u is offered the choice of S_{u,f_u} or $\bar{S}_{u,f_u} \cup \mathrm{LO}_u$. If u prefers the former, then it gets S_{u,f_u}, and f_u gets $\bar{S}_{u,f_u} \cup \mathrm{LO}_u$. In this case, f_u may envy u, but this is EFX. If u prefers $\bar{S}_{u,f_u} \cup \mathrm{LO}_u$, it gets this bundle, and f_u gets $S_{u,f_u} (= T_{u,f_u})$. Again, in this case, there is no envy.

We note the following properties of the algorithm. In each iteration of the outer for loop, a vertex $u \in L$ is chosen, and the goods in St_u are assigned to the agents in St_u. The allocation to all other agents remains unchanged. Since St_u does not contain any agent from L other than u, an agent $u' \in L$ has no allocated goods until it is resolved, and the allocation to agent u' is not changed after it is resolved. Finally, no goods are removed from an agent once assigned.

▶ **Theorem 3.** *Algorithm 1 runs in polynomial time, and returns an EFX allocation.*

The polynomial running time is easily seen, and is because the procedure $\textsc{Cut}(u, S)$ runs in polynomial time for cancelable valuations. The EFX property follows immediately from the next lemma.

Algorithm 1 Bipartite-EFX.

Require: Bipartite multi-graph $G = (L \cup R, E)$, and vertices with cancelable valuations over E.
Ensure: EFX allocation A.

Initially, for each vertex u, $A_u = \emptyset$
for each vertex $u \in L$ **do**
  for each vertex $w \in N_u$ **do**
    $(E^1_{u,w}, E^2_{u,w}) = \textsc{Cut}(w, E_{u,w})$. ▶ Returns an EFX partition for identical agents
    $S_{u,w} = \arg\max_{S \in \{E^1_{u,w}, E^2_{u,w}\}} v_u(S)$
    $T_{u,w} = \arg\max_{S \in \{E^1_{u,w}, E^2_{u,w}\}} v_w(S)$. ▶ Preferred bundles for u and w from $E_{u,w}$
  $f_u = \arg\max_{w \in N_u} v_u(S_{u,w})$
  $\mathrm{LO}_u = \bigcup_{w \in N_u \setminus \{f_u\}} \bar{T}_{u,w}$ ▶ Left-over bundles from all ordinary neigbours
  $A_w = A_w \cup T_{u,w}$ for each vertex $w \in N_u \setminus \{f_u\}$ ▶ Ordinary neighbours get their preferred bundle
  if $(S_{u,f_u} = T_{u,f_u})$ **then** ▶ Both u and f_u prefer the same bundle in E_{u,f_u}
    if $(v_u(S_{u,f_u}) > v_u(\bar{S}_{u,f_u} \cup \mathrm{LO}_u))$ **then** ▶ u prefers S_{u,f_u} over all the left-over bundles
      $A_u = S_{u,f_u}$
      $A_{f_u} = A_{f_u} \cup \bar{S}_{u,f_u} \cup \mathrm{LO}_u$
    else ▶ u prefers the left-over bundles
      $A_u = \bar{S}_{u,f_u} \cup \mathrm{LO}_u$
      $A_{f_u} = A_{f_u} \cup S_{u,f_u}$
  else ▶ u and f_u prefer different bundles
    $A_u = S_{u,f_u} \cup \mathrm{LO}_u$
    $A_{f_u} = A_{f_u} \cup T_{u,f_u}$
return A

▶ **Lemma 4.** *Fix an agent $u \in L$. Let A be the partial allocation after resolving St_u. Then the only possible envy in allocation A is that a resolved agent $u' \in L$ is envied by its favourite agent $f_{u'}$. Further, the partial allocation A is EFX.*

Proof. The proof is by induction on the iterations of the outer for loop. The claim is clearly true for the initial empty allocation. Let $\hat{A}$ be the allocation before u is resolved. Then by the induction hypothesis, since u is not yet resolved in $\hat{A}$, no agents in St_u are envied.

In the current iteration, when St_u is resolved, only agents in St_u get goods, while agents not in St_u retain their allocation. Hence, any new envy edges must be to agents in St_u. Further since only agents in St_u value the goods in St_u, any new envy edges must also be from agents in St_u.

Suppose agent w envies agent $w' \notin \mathrm{St}_u$ in $\hat{A}$ (as established, agents in St_u are not envied in $\hat{A}$). Thus $v_w(\hat{A}_w) < v_w(\hat{A}_{w'})$. In the current iteration, the allocation to w' does not change, while w does not lose any goods. Hence any such envy remains EFX. Thus to prove the lemma, we only need to show that any new envy between agents in St_u is from f_u to u, and is EFX.

Consider first an ordinary neighbour w. Of the goods allocated in this iteration, w only values $E_{u,w}$. Of this set, it gets its preferred partition $T_{u,w}$. Then since w did not envy anyone in St_u earlier, for any agent $z \in \mathrm{St}_u$, $v_w(\hat{A}_w) \geq v_w(\hat{A}_z)$. Then

$$v_w(A_w) = v_w(\hat{A}_w \cup T_{u,w}) \geq v_w(\hat{A}_z \cup \bar{T}_{u,w}) \geq v_w(A_z)\,.$$

The first inequality is because valuations are cancelable, and w prefers $\hat{A}_w$ to $\hat{A}_z$, and $T_{u,w}$ to $\bar{T}_{u,w}$. The second inequality is because in this iteration, any agent other than w either does not receive any good valued by w, or receives the bundle $\bar{T}_{u,w}$.

Consider f_u, agent u's favourite neighbour. Again of the goods allocated in this iteration, f_u is only interested in E_{u,f_u}. Of this set, it either gets its preferred bundle T_{u,f_u} (in either Line 14 or in Line 17), or it gets the bundle $\bar{S}_{u,f_u}$ in Line 11. In the former case, since the valuations are cancelable and agent f_u did not earlier envy any agent in St_u, agent f_u does not envy any agent in St_u in allocation A either. In the latter case, f_u gets the bundle $\bar{S}_{u,f_u}$, and hence may envy u since agent u gets $S_{u,f_u} = T_{u,f_u}$. But note that then $A_u = S_{u,f_u}$, and this is an EFX partition since f_u cuts the set of items E_{u,f_u}. Thus, the only possible new envy edge from f_u is to u, and this is EFX.

Lastly, consider agent u. We show that u will not envy any agent in St_u. If u gets $S_{u,f_u} \cup \mathrm{LO}_u$ (Line 17), then by definition S_{u,f_u} is preferred over all the other bundles allocated to the other agents in St_u, and u additionally gets all the left-over bundles. Otherwise, u gets a choice between S_{u,f_u} and $\bar{S}_{u,f_u} \cup \mathrm{LO}_u$; the other bundle is given to f_u. Then clearly u does not envy f_u. The bundle S_{u,f_u} has a higher value for u than $T_{u,w}$ for any neighbour w. Since u chooses the bundle with the highest value, u does not envy any ordinary neighbour either. ◀

4 EFX Allocations for Monotone Valuations in Tree Multi-graphs

We now show that for monotone valuations, an EFX allocation exists in tree multi-graphs. Note that even for two agents with monotone submodular valuations, computing an EFX allocation is PLS-complete [19], and hence a polynomial-time algorithm for tree multi-graphs may not exist. Since trees are also bipartite, a polynomial time algorithm for *cancelable* valuations follows from the previous section.

Our algorithm for trees is recursive and again utilizes the procedure $\textsc{Cut}(v, S)$, that obtains an EFX partition (S^1, S^2) of S for two identical agents with valuation v. Let $T = (V, E)$ be the tree multi-graph, and ℓ be a leaf with parent p. Then $E_{\ell,p}$ are the edges between ℓ and p, and no other agent values these goods. Inductively, let A' be an EFX allocation of goods $E \setminus E_{\ell,p}$ to agents $V \setminus \{\ell\}$. Our objective is then to extend A' to an EFX allocation of goods E.

For this, let (S^1, S^2) be the partition obtained when p cuts $E_{\ell,p}$. We define $A_\ell = \arg\max_{S \in \{S^1, S^2\}} v_\ell(S)$ and $\bar{A}_\ell = E_{\ell,p} \setminus A_\ell$. Note that ties are broken arbitrarily, ensuring that A_ℓ is a single bundle. Agent ℓ gets its preferred bundle A_ℓ. If p is not envied in allocation A', then it gets $\bar{A}_\ell$, and the algorithm terminates. If p is already envied in allocation A', it cannot get additional goods. Let s be a source in the envy graph for A' with a path to p. We add $\bar{A}_\ell$ to agent s's allocation. If p does not envy s now, the algorithm terminates. Else, there is an envy cycle where p envies s. Resolving the envy cycle gives us an EFX allocation.

▶ **Theorem 5.** *Given a tree multi-graph, $T = (V, E)$ with monotone valuations, Algorithm Tree-EFX returns an EFX allocation A.*

Proof. By induction, assume that A' is an EFX allocation on $T' = (V \setminus \{\ell\}, E \setminus E_{\ell,p})$. Note that given an EFX allocation, resolving envy cycles does not destroy this property, since each agent's value does not decrease, and bundles are merely reassigned. Hence, the partial allocation in Line 11 is EFX for agents $V \setminus \{\ell\}$, and $A_\ell = \emptyset$.

On allocating A_ℓ to agent ℓ, clearly ℓ does not envy any agent. The only agent that could envy ℓ is agent p, since only agents p and ℓ value the bundle A_ℓ. Further, since p gets to cut the bundle $E_{\ell,p}$, as long as p possesses a bundle with value at least $v_p(\bar{A}_\ell)$, its envy towards ℓ will be EFX.

Algorithm 2 Tree-EFX.

Require: A tree multi-graph $T = (V, E)$ with $|V| \geq 2$ and monotone valuations.
Ensure: An EFX allocation A

1: **if** $(|V| = 2)$ **then**
2: Let $V = \{\ell, p\}$
3: $(S^1, S^2) = \textsc{Cut}(p, E_{\ell,p})$
4: $A_\ell = \arg\max_{S \in \{S^1, S^2\}} v_\ell(S)$, $A_p = \bar{A}_\ell$
5: **return** $A = (A_\ell, A_p)$
6: Let ℓ be a leaf in T
7: $A' = \text{Tree-EFX}(V \setminus \{\ell\}, E \setminus E_{\ell,p})$ ▶ Recursive call on the smaller tree without leaf ℓ
8: $G_{A'}$ is the envy graph for allocation A'
9: **while** (there is an envy-cycle C in $G_{A'}$) **do** ▶ Resolve any envy-cycles
10: $A' = A'^C$
11: $A_u = A'_u$ for all vertices $u \neq \ell$
12: $(S^1, S^2) = \textsc{Cut}(p, E_{\ell,p})$ ▶ p cuts the bundle $E_{\ell,p}$
13: $A_\ell = \arg\max_{S \in \{S^1, S^2\}} v_\ell(S)$ ▶ Leaf ℓ gets its preferred bundle
14: **if** (p is a source in $G_{A'}$) **then**
15: $A_p = A_p \cup \bar{A}_\ell$ ▶ If p is not envied, p gets the left-over items
16: **else**
17: Let s be a source with a path P to p. ▶ Else, assign left-over items to s, resolve any envy cycle.
18: $A_s = A_s \cup \bar{A}_\ell$
19: **if** $(v_p(A_p) < v_p(A_s))$ **then**
20: Let C be the envy-cycle consisting of the edge (p, s) and the s-p path P.
21: $A = A^C$
22: **return** A

If p is a source in the envy graph, then the condition in Line 14 is satisfied and gets the bundle $\bar{A}_\ell$. No agent will envy p, since agents $V \setminus \{p, \ell\}$ do not value the bundle $\bar{A}_\ell$, and ℓ already gets its preferred bundle. Thus the allocation A is EFX.

If p is not a source, then since the envy-graph is acyclic (and ℓ does not envy any agent), there is a source s with a path to p in the envy graph. Let P be this path. We assign $\bar{A}_\ell$ to agent s. The only agent that could now envy s is p. If p envies s, there is an envy cycle, which we resolve. We claim that the resulting allocation is EFX. This is because agent ℓ continues to not envy any agent. Agent p's value for its bundle is now at least $v_p(\bar{A}_\ell)$, and hence any envy towards agent ℓ is EFX. Further agent p's value for its bundle is also at least its earlier value; since resolving the cycle only shifted bundles around, any envy towards agents $V \setminus \{\ell\}$ is also EFX by induction. Finally, for other agents, their value does not decrease, and any envy is unaffected by the shifting of bundles. Hence the allocation is EFX. ◀

5 EFX Allocations in t-Chromatic Multi-graphs with Girth at Least $(2t - 1)$

We now extend our algorithm and analysis to show that EFX allocations exist for agents with cancelable valuations in multi-graphs with girth at least $2t - 1$, where t is the chromatic number of the graph. A multi-graph is t-chromatic if t is the minimum number of colours

required to colour the vertices, so no edge has both end-points the same colour. The girth of a multi-graph is the length of the shortest simple cycle, without using parallel edges. For example, bipartite multi-graphs are 2-chromatic with girth at least 4, while the Petersen graph (with parallel edges) is 3-chromatic with girth 5. We will assume that we are given a t-colouring of the multi-graph.

Our algorithm proceeds as follows. Given a multi-graph $G = (V, E)$ coloured with t colours, let $c(u) \in [t]$ be the colour of each vertex $u \in V$. Let $C^i = \{u : c(u) = i\}$ be the independent set consisting of all vertices coloured i. We think of the vertices as ordered from left to right by their colour, and say a vertex u is *to the left of* w if $c(u) < c(w)$. For $i = 1, \ldots, t-1$, define $L^i := C^i$, and $R^i := C^{i+1} \cup \ldots \cup C^t$.

We proceed in phases. In phase i, we consider the bipartite multi-graph $G^i = (V^i = L^i \cup R^i, E^i)$ where an edge $e \in E$ is in E^i if $|r(e) \cap L^i| = |r(e) \cap R^i| = 1$, i.e., each edge is from a vertex in L^i to a vertex in R^i. Note that each vertex $u \in V \setminus C^t$ appears in a left bipartition in exactly one phase, when we consider all edges to vertices to the right of u. Thus each edge appears in exactly one phase.

Now in phase i, we run Algorithm 1 on the bipartite multi-graph G^i. The only change we make is that in Algorithm 1, initially all vertices – in particular, vertices $u \in L$ – had empty allocations. This may not be the case in the current algorithm in the second phase onwards. Hence, we remove Line 1 from Algorithm 1, which initializes the allocation, within the outer for loop. Further in Lines 16 and 19 in Algorithm 3, agent $u \in L^i$ retains its earlier allocation. In Line 13, when $u \in L^i$ is possibly envied by f_u, its earlier allocation passes to f_u. Algorithm 3 formally states the algorithm.

A *phase* is an iteration of the outer loop. In phase i, the vertices in C^i are picked sequentially in the inner for loop.

Structures

For our analysis, we slightly need to redefine *structures*. For a vertex $u \in C^i$, we now define the structure St_u as the subgraph induced by $u \cup (N_u \cap R^i)$. That is, a structure now consists of the subgraph induced by u and all neighbours *to its right*. As before, u is the *root* of the structure. Now each iteration in phase i of our algorithm resolves the structure St_u for some $u \in C^i$, and thus assigns all the edges in St_u. A vertex u is resolved if St_u is resolved.

Fix agent $u \in C^i$. As before, for each $w \in (N_u \cap R^i)$, $(E^1_{u,w}, E^2_{u,w}) = \textsc{Cut}(w, E_{u,w})$ is the partition of $E_{u,w}$ returned when agent w cuts. Define $S_{u,w} := \arg\max\{v_u(E^1_{u,w}), v_u(E^2_{u,w})\}$ as the bundle preferred by u, and $T_{u,w} := \arg\max\{v_w(E^1_{u,w}), v_w(E^2_{u,w})\}$ as the bundle preferred by w. As earlier, ties are broken arbitrarily to ensure that $S_{u,w}$ and $T_{u,w}$ are single bundles. If either agent is indifferent between the two bundles, we break ties so that $S_{u,w} \neq T_{u,w}$. Further, for $S \subseteq E_{u,w}$, we define $\bar{S} = E_{u,w} \setminus S$. Then u's *favourite* neighbour $f_u := \arg\max_{w \in N_u \cap R_i} v_u(S_{u,w})$, with ties broken arbitrarily.
This is the right neighbour that offers u the highest-value bundle after cutting the adjacent edges. The other right neighbours are simply called *ordinary* neighbours.

We note the following properties of our algorithm. In phase i of the algorithm, the inner for loop picks an agent $u \in C^i$, and resolves St_u. The goods in St_u are assigned to the agents in St_u. The allocation to other agents is unchanged. Hence in particular, the allocation to a resolved agent does not change subsequently. Furthermore, while St_u is resolved, agents in St_u do not lose any goods, except possibly u itself. Agent u may however give its entire set of goods to its favourite neighbour f_u (Line 13). However, in this case, agent u gets S_{u,f_u}, which it prefers over its earlier allocation and all the left-over goods. Thus, the value of each agent for its bundle is nondecreasing.

Algorithm 3 t-Chromatic-EFX.

Require: t-Chromatic multi-graph $G = (L \cup R, E)$ with girth $(2t-1)$ and cancelable valuations. C^i is the set of vertices coloured $i \in [t]$.

Ensure: EFX allocation A.

1: Initially, for each vertex u, $A_u = \emptyset$. Resolved agents $\text{Res} = \emptyset$.
2: **for** $i = 1$ to $t-1$ **do**
3: $L^i = C^i$, $R^i = C^{i+1} \cup \ldots \cup C^t$. ▶ Only consider edges between L^i, R^i
4: **for** each vertex $u \in L^i$ **do**
5: **for** each vertex $w \in N_u \cap R^i$ **do**
6: $(E^1_{u,w}, E^2_{u,w}) = \textsc{Cut}(w, E_{u,w})$. ▶ Right vertex cuts
7: $S_{u,w} = \arg\max_{S \in \{E^1_{u,w}, E^2_{u,w}\}} v_u(S)$
8: $T_{u,w} = \arg\max_{S \in \{E^1_{u,w}, E^2_{u,w}\}} v_w(S)$.
9: $f_u = \arg\max_{w \in N_u \cap R^i} v_u(S_{u,w})$
10: $\text{LO}_u = \bigcup_{w \in N_u \setminus \{f_u\} \cap R^i} \bar{T}_{u,w}$ ▶ Left-over bundles from all ordinary neigbours
11: $A_w = A_w \cup T_{u,w}$ for all $w \in N_u \setminus \{f_u\} \cap R^i$ ▶ Ordinary neighbours get their preferred bundle
12: **if** $(S_{u,f_u} = T_{u,f_u})$ **then** ▶ Both u and f_u prefer the same bundle in E_{u,f_u}
13: **if** $(v_u(S_{u,f_u}) > v_u(A_u \cup \bar{S}_{u,f_u} \cup \text{LO}_u))$ **then** ▶ u prefers S_{u,f_u} over left-over and its own goods
14: $A_{f_u} = A_{f_u} \cup A_u \cup \bar{S}_{u,f_u} \cup \text{LO}_u$
15: $A_u = S_{u,f_u}$
16: **else** ▶ u prefers the left-over bundles
17: $A_u = A_u \cup \bar{S}_{u,f_u} \cup \text{LO}_u$
18: $A_{f_u} = A_{f_u} \cup S_{u,f_u}$
19: **else** ▶ u and f_u prefer different bundles
20: $A_u = A_u \cup S_{u,f_u} \cup \text{LO}_u$
21: $A_{f_u} = A_{f_u} \cup T_{u,f_u}$
22: $\text{Res} = \text{Res} \cup \{u\}$ ▶ Bookkeeping: u is resolved.
23: **return** A

Much of our effort is towards proving Lemma 11, which shows that any envy is within a structure, and is from a favourite neighbour f_u to the root u. In particular, we want to avoid the following. Say for some interim allocation A, agent u' values goods in both A_u and A_{f_u}. Then if A_u is transferred to f_u in Line 13, agent u' should not envy f_u.

We show this by contradiction: that if agent u' does indeed start envying f_u, then it must have short paths to both u and f_u, and hence, taken with any edge between u and f_u, there is a cycle of length less than $2t-1$. We do this in two steps: Claim 7 shows that if u' envies the bundle of goods $A_u \cup A_{f_u}$, then these must contain a good from $\text{St}_{u'}$. Claims 9 and 10 then establish bounds on the distance from u' for u and f_u.

Next, we state the claims required to prove Lemma 11. Our first claim is that an unresolved agent does not envy the *entire* set of goods held by all other unresolved vertices.

▷ Claim 6. Let A and Res be the allocation and the set of resolved vertices at Line 5. Let z be an unresolved agent. Then

$$v_z(A_z) \geq v_z\left(\bigcup_{w \notin \text{Res},\, w \neq z} A_w\right).$$

Proof. Fix an unresolved agent z. We prove this by induction on the iterations of the inner loop. Initially, the allocation is empty, and the claim is trivially satisfied. Assume this is true prior to the loop when a vertex u is resolved. Let $\hat{A}$, $\widehat{\text{Res}}$ be the allocation and set of resolved vertices before u is resolved, and A, $\text{Res} = \widehat{\text{Res}} \cup \{u\}$ be the allocation and set of resolved vertices after. Note that the allocation to resolved agents does not change subsequently. Hence if z now envies the set of goods held by unresolved agents, new goods that z values must be allocated to unresolved agents.

In this loop, only the allocation to agents in St_u is modified. Further, none of the agents in St_u are resolved prior to the loop, i.e., $\text{St}_u \cap \widehat{\text{Res}} = \emptyset$. Hence $v_z(\hat{A}_z) \geq v_z\left(\cup_{u' \in \text{St}_u,\, u' \neq z} \hat{A}_{u'}\right)$ by the induction hypothesis. Now if $z \notin \text{St}_u$, then no additional goods it values are allocated in this loop. Hence the claim holds by induction. Thus, since $z \neq u$ (since z is unresolved after the loop), it must be that z is a neighbour to the right of u. Of the goods allocated in this loop, z only values $E_{u,z}$.

If z is an ordinary neighbour, then z additionally gets its preferred bundle $T_{u,z}$, while the bundle $\bar{T}_{u,z}$ is allocated to other agents. Hence, since valuations are cancelable, the claim remains true. On the other hand, if $z = f_u$, then z could lose its preferred bundle $T_{u,z}$ to agent u. But in this case agent u is resolved, hence $u \notin \text{Res}$. Since z additionally gets $\bar{T}_{u,z}$ and does not lose any goods, the claim remains true after the loop also. ◁

The next claim builds on the previous claim to show that if any vertex z envies the union of goods held by some set S of unresolved agents, then z must be resolved, and some agent in S must hold a good from St_z.

▷ Claim 7. Let A and Res be the allocation and the set of resolved vertices at Line 5. Let z be any agent, and S is a subset of unresolved vertices so that

$$v_z(A_z) < v_z\left(\bigcup_{w \in S, z \notin S} A_w\right).$$

Then z is resolved, and for some agent $w \in S$, A_w contains some good $g \in \text{St}_z$.

Proof. By Claim 6, if z is unresolved, then $v_z(A_z) \geq v_z\left(\cup_{w \in S,\, w \neq z} A_w\right)$. Hence z must in fact be resolved. We prove the contrapositive by induction, that if a subset of unresolved agents S do not hold any good from St_z, then $v_z(A_z) \geq v_z(\cup_{w \in S} A_w)$. Clearly, the claim holds in any iteration before z is resolved.

Consider the iteration where z is resolved. Every agent in St_z receives goods from St_z, hence, we only consider subsets S consisting of agents not in St_z. But the allocation for these agents does not change, and z's value for its bundle is nondecreasing, hence the claim holds in this iteration.

Consider any later iteration, where an agent $u \neq z$ is resolved. Then $u \notin S$, since u is resolved. Let $\hat{A}$ be the allocation before resolving St_u, and A the allocation after. Again, only the allocation for agents in St_u changes, hence by induction we only need to consider subsets S that contain agents in St_u. Note that since agent z is already resolved, z does not value any goods in St_u. Hence the allocation of goods in St_u does not affect the right hand side of the inequality in the claim.

Other than the allocation of goods in St_u, the only change that happens in the algorithm is that the prior bundle $\hat{A}_u$ may be transferred to f_u. Thus if $\hat{A}_u$ is not transferred to f_u, or $f_u \notin S$, then z's value for the aggregate bundle $\cup_{w \in S} A_w$ is unchanged, and the claim holds.

Thus, $f_u \in S$, $u \notin S$, and the only relevant change that occurs in the iteration is that the bundle $\hat{A}_u$ is transferred to agent f_u. Note that since $f_u \in S$, f_u does not hold any goods from St_z, and hence $\hat{A}_u$ does not contain any goods from St_z. Now in the allocation $\hat{A}$, consider the subset $\hat{S} = S \cup \{u\}$. Then

$$v_z(A_z) = v_z(\hat{A}_z) \geq v_z\left(\bigcup_{w \in \hat{S}} \hat{A}_w\right) \quad \text{(by the induction hypothesis)}$$
$$= v_z\left(\hat{A}_u \cup \bigcup_{w \in S} \hat{A}_w\right) = v_z\left(\bigcup_{w \in S} A_w\right),$$

proving the claim. ◁

▶ **Proposition 8.** *For an agent u, let $\hat{A}$ and A be the allocations before and after St_u is resolved. If for some agent z and good g, $g \in \hat{A}_z$ but $g \notin A_z$, then $z = u$ and $g \in A_{f_u}$. Thus in a phase, any good allocated before the phase began can only be transferred from the root of a structure to its favourite neighbour.*

Proof. As noted, when St_u is resolved, the goods in the structure are allocated to the agents in the structure. If Line 13 is executed, then additionally the bundle A_u is transferred to agent f_u (and agent u gets the bundle S_{u,f_u}). Thus the only way that a good that is assigned to an agent before St_u is resolved can be moved, is if it was assigned to u and then is transferred to agent f_u. The proposition follows.

Since a root only appears once in a phase, a good once transferred in a phase cannot be transferred again, and hence any good allocated prior to a phase can only be transferred once, from the root of a structure to its favourite neighbour. ◀

The next few results show that a good valued by an agent z cannot be very far from z.

▷ Claim 9. Let A be the allocation at Line 5. If agent z values a good g and $g \in A_w$, then $\mathrm{dist}(z, w) \leq c(w)$ along allocated edges.

Proof. By Proposition 8, a good can only be transferred rightward, from an agent being resolved to its favourite neighbour. If $c(w) = 1$, then once St_w is resolved, w's allocation does not change. Thus if $g \in A_w$ and $c(w) = 1$, w received this good in phase 1 when St_w was resolved. Since z values g, z must be in N_w, and there is a path of length 1 between z and w along the edge set $E_{w,z}$ satisfying the claim.

Now suppose $c(w) \geq 2$. Consider the phase when g is initially allocated, say to an agent w' when St_u is resolved. Since z values g, $z \in \mathrm{St}_u$. Hence z and w' have a path of length 2 along the edges of St_u. From Proposition 8, in each subsequent phase, g is transferred from the root of a structure to its favourite neighbour. Suppose g is allocated to w in phase k. Then w must be a right neighbour for the agent being resolved, and hence the colour $c(w) \geq k + 1$. There are at most $k - 1$ phases after the phase g is initially allocated. Thus there is a path of length $k - 1 + 2 \leq c(w)$ along allocated edges from w to z. ◁

If z not only values g, but $g \in \mathrm{St}_z$ and is held by agent w, then we can get a tighter bound on $\mathrm{dist}(z, w)$.

▷ Claim 10. Let A be the allocation at Line 5. Let there be agents z, w, and good $g \in \mathrm{St}_z \cap A_w$. Then $\mathrm{dist}(z, w) \leq c(w) - c(z)$ along allocated edges.

Proof. Since $g \in \mathrm{St}_u$, when St_u is resolved, it is assigned either to u or to a neighbour to the right. Further any reassignment of g transfers it to a neighbour further to the right. Hence if $u \neq w$, then w must be to the right of u, hence $c(w) > c(u)$.

Now when g is initially allocated in phase $c(u)$, it is allocated to an agent w' at a distance 1 from u along the edges $E_{u,w'}$. Suppose g is allocated to agent w in phase k. Then $c(w) > k$, and there are at most $k - c(u)$ phases after the phase when g is initially allocated. Thus there is a path of length $k - c(u) + 1 \leq c(w) - c(u)$ along allocated edges from u to w. ◁

We now prove our main lemma.

▶ **Lemma 11.** *Let A be the allocation at Line 5. Then if w envies u, then u is a resolved vertex, and $w = f_u$ (i.e., w is u's favourite vertex in St_u).*

Proof. We will prove the lemma by induction. For the base case, the initial allocation is empty and trivially satisfies the two properties. Now fix any vertex u. Let $\hat{A}$ be the allocation before the structure rooted at u is resolved, and A be the allocation after resolving St_u.

None of the vertices in St_u are resolved in $\hat{A}$, and hence by the induction hypothesis, none of these are envied in $\hat{A}$. When St_u is resolved, only the allocation to agents in St_u is modified, while the allocation to other agents is unchanged. The value of agents in St_u does not decrease. Hence, any new envy edge must be towards agents in St_u.

Consider an ordinary neighbour w in St_u. Agent w is not envied in $\hat{A}$. It receives its preferred bundle $T_{u,w}$; this is only valued by u and w, hence no agent other than u will envy w. To see that u also will not envy w, we first claim that $v_u(\hat{A}_w) = 0$, i.e., u does not value w's bundle in $\hat{A}$. To prove this, assume for a contradiction that u values w's bundle. Then by Claim 9, there exists a u-w path along allocated edges of length at most t. Including the (unallocated) edges $E_{u,w}$, this gives a cycle of length $t + 1$. If $t = 2$, this gives us a 3-cycle, which would mean that the graph is not 2-colourable. If $t > 2$, then $t + 1 < 2t - 1$, but the graph has girth at least $2t - 1$. In either case we get a contradiction. Thus, $v_u(\hat{A}_w) = 0$.

Further, $v_u(A_u) \geq v_u(T_{u,w})$. This is because agent u prefers S_{u,f_u} over $T_{u,w}$, and always has the choice of keeping S_{u,f_u}. Hence, it ends up with a bundle of value at least that of $T_{u,w}$. Then since $v_u(\emptyset) \geq v_u(\hat{A}_w)$ and $v_u(A_u) \geq v_u(T_{u,w})$, since valuations are cancelable, $v_u(A_u) \geq v_u(A_w)$. Thus, an ordinary neighbour $w \in \mathrm{St}_u$ is not envied.

Now consider agent u. As shown, agent u is not envied in $\hat{A}$. While resolving St_u, the only goods agent u can possibly receive are goods in St_u, and hence no agent outside St_u will envy u. Within St_u, all ordinary neighbours receive their preferred bundle, and hence will not envy u. Thus the only agent that could possibly envy u after resolving St_u is f_u.

Now if f_u gets its preferred bundle T_{u,f_u} (in Lines 16 and 19), it does not envy u. If agent f_u gets $\bar{T}_{u,f_u}$ (in Line 13), then this is an EFX partition for f_u (since f_u cut sE_{u,f_u}), and agent u only gets T_{u,f_u}. In this case, f_u envies u, but the envy is EFX.

Lastly, consider agent f_u. This case is more complicated than the earlier cases, since the bundle $\hat{A}_u$ previously held by u could be transferred to f_u in Line 13. We will show however that no agent envies f_u.

First, let's consider agents in St_u. Again, the ordinary neighbours w did not envy f_u earlier, and from the goods $E_{u,w}$ they get their preferred bundle $T_{u,w}$. Hence, since valuations are cancelable, they will not envy f_u.

Agent u, as argued previously, by Claim 9, does not value $\hat{A}_{f_u}$, else there is a cycle of length strictly smaller than $2t - 1$. Further agent u gets a bundle of value at least that of S_{u,f_u}, and hence will not envy f_u.

For the last case, consider any agent $u' \notin \mathrm{St}_u$. In allocation $\hat{A}$, u' does not envy u or f_u. Since u' does not value goods in E_{u,f_u}, if it envies A_{f_u}, clearly (i) it must value both $\hat{A}_u$ and $\hat{A}_{f_u}$, and (ii) it must envy $\hat{A}_u \cup \hat{A}_{f_u}$.

Suppose $c(u') \geq 2$. Then from Claim 7, either u or f_u holds a good from $\mathrm{St}_{u'}$ (say u). By Claim 10, u is then at distance $c(u) - c(u')$ from u' along allocated edges. From (i) in the previous paragraph, and Claim 9, f_u is at distance at most $c(f_u)$ from u' along allocated edges. Thus, there is u-f_u path along allocated edges (via u') of length $c(f_u) + c(u) - c(u')$ $\leq 2t-3$ (since $c(u) \leq t-1$ and $c(u') \geq 2$). But then including the (unallocated) edges E_{u,f_u}, there is a cycle of length $2t-2$, which is a contradiction, since we assume the graph has girth at least $2t-1$.

Finally, suppose $c(u') = 1$. Then note that all the goods that u' values are in $\mathrm{St}_{u'}$. Since both u and f_u possess goods that u' values, by Claim 10, they are at distance $c(u) - 1$ and $c(f_u) - 1$ respectively from u' along allocated edges. Since $c(u) \leq t-1$, there is a path of length $2t-3$ from u to f_u (via u') along allocated edges. But then including the (unallocated) edges E_{u,f_u}, there is a cycle of length $2t-2$, which is again a contradiction. Thus, agent f_u is not envied in the allocation A. This completes the proof. ◀

We now can easily prove that the allocation obtained is EFX.

▶ **Theorem 12.** *Algorithm 3 obtains an EFX allocation for agents with cancelable valuations in polynomial time.*

Proof. The proof for the running time for Algorithm 3 is obtained easily, since for cancelable valuations $\textsc{Cut}(u, S)$ runs in polynomial time, and the other steps are straightforward.

We now show that the algorithm obtains an EFX allocation. As before, the proof is by induction on the iterations of the inner for loop. For the base case, the statement clearly holds for the empty allocation. Then consider an agent u, and let $\hat{A}$ and A be the allocations before and after St_u is resolved.

Consider first an envy edge from agent w to agent u' in allocation $\hat{A}$. We will show that this envy remains EFX in allocation A as well. By Lemma 11, agent u' must be resolved. No agent in St_u is resolved in $\hat{A}$, hence $u' \notin \mathrm{St}_u$. Since only agents in St_u have their allocations modified, agent u''s allocation is not modified. Agent w is either also not in St_u (in which case its allocation also does not change), or is in St_u (in which case its value does not decrease). In either case, either w does not envy u' in A, or the envy remains EFX.

By Lemma 11, the only envy edge that may be in A but not in A' is from f_u to u. This happens in Line 11, when u gets S_{u,f_u}. But then agent f_u gets $\bar{S}_{u,f_u}$. Since agent f_u cuts the bundle E_{u,f_u}, and agent u has no goods besides S_{u,f_u}, the envy is EFX. ◀

Finally, we show that EFX allocations exist in cycle multi-graphs, from the earlier results.

▶ **Theorem 13.** *For agents with cancelable valuations in a cycle multi-graph, EFX allocations exist.*

Proof. We consider three separate cases, depending on the length of the cycle. For even-length cycles, the result follows from Theorem 3, since such a cycle is a bipartite multi-graph. If the length is 3, then the result follows from EFX existence for three agents [16]. Finally, if the cycle has odd length of at least 5, the chromatic number is 3, and the existence follows from Theorem 12, setting $t = 3$, since the girth is least 5. ◀

6 Conclusion

Our paper extends the work on EFX existence to important classes of multi-graphs. We view our results as a significant extension over the work by Christodoulou et al. [18] since their results are for simple graphs. Our work also shows that the well-known cut-and-choose paradigm can be leveraged to obtain results in fairly general settings. Our algorithm is another tool in the toolbox for EFX allocations, different, for example, from the multiple-phase algorithms used in many other papers.

We note that if edges are chores instead of good, i.e., an edge is negatively-valued by both adjacent vertices, an envy-free allocation can be obtained by assigning each edge to a vertex other than its endpoints. The existence of EFX allocations for general multi-graphs with positively valued edges remains an open problem. We also find it interesting that the case of a cycle multi-graph with 3 agents does not yet have a simpler algorithm than for general 3 agents. Coming up with a simpler algorithm here may be crucial to extending our work to general multi-graphs.

References

1 Mahyar Afshinmehr, Alireza Danaei, Mehrafarin Kazemi, Kurt Mehlhorn, and Nidhi Rathi. EFX allocations and orientations on bipartite multi-graphs: A complete picture. *CoRR*, abs/2410.17002, 2024. doi:10.48550/arXiv.2410.17002.

2 Mahyar Afshinmehr, Alireza Danaei, Mehrafarin Kazemi, Kurt Mehlhorn, and Nidhi Rathi. EFX allocations and orientations on bipartite multi-graphs: A complete picture. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 32–40, 2025. doi:10.5555/3709347.3743514.

3 Hannaneh Akrami, Noga Alon, Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, and Ruta Mehta. EFX: A simpler approach and an (almost) optimal guarantee via rainbow cycle number. In *Proceedings of the 24th ACM Conference on Economics and Computation, EC 2023, London, United Kingdom, July 9-12, 2023*, page 61. ACM, 2023. doi:10.1145/3580507.3597799.

4 Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artificial Intelligence*, 322:103965, 2023. doi:10.1016/J.ARTINT.2023.103965.

5 Georgios Amanatidis, Aris Filos-Ratsikas, and Alkmini Sgouritsa. Pushing the frontier on approximate EFX allocations. In Dirk Bergemann, Robert Kleinberg, and Daniela Sabán, editors, *Proceedings of the 25th ACM Conference on Economics and Computation, EC 2024, New Haven, CT, USA, July 8-11, 2024*, pages 1268–1286. ACM, 2024. doi:10.1145/3670865.3673582.

6 Georgios Amanatidis, Evangelos Markakis, and Apostolos Ntokos. Multiple birds with one stone: Beating 1/2 for efx and gmms via envy cycle elimination. *Theoretical Computer Science*, 841:94–109, 2020. doi:10.1016/J.TCS.2020.07.006.

7 Haris Aziz, Ioannis Caragiannis, Ayumi Igarashi, and Toby Walsh. Fair allocation of indivisible goods and chores. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 53–59, 2019. doi:10.24963/IJCAI.2019/8.

8 Siddharth Barman, Umang Bhaskar, Yeshwant Pandit, and Soumyajit Pyne. Nearly equitable allocations beyond additivity and monotonicity. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 9494–9501. AAAI Press, 2024. doi:10.1609/AAAI.V38I9.28804.

9 Kristóf Bérczi, Erika R. Bérczi-Kovács, Endre Boros, Fekadu Tolessa Gedefa, Naoyuki Kamiyama, Telikepalli Kavitha, Yusuke Kobayashi, and Kazuhisa Makino. Envy-free relaxations for goods, chores, and mixed items. *Theor. Comput. Sci.*, 1002:114596, 2024. doi:10.1016/J.TCS.2024.114596.

10 Ben Berger, Avi Cohen, Michal Feldman, and Amos Fiat. Almost full EFX exists for four agents. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, February 22 - March 1, 2022*, pages 4826–4833. AAAI Press, 2022. doi:10.1609/AAAI.V36I5.20410.

11 Umang Bhaskar, Gunjan Kumar, Yeshwant Pandit, and Rakshitha. Towards envy-freeness relaxations for general nonmonotone valuations. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 298–306, 2025. doi:10.5555/3709347.3743543.

12 Umang Bhaskar, A. R. Sricharan, and Rohit Vaish. On approximate envy-freeness for indivisible chores and mixed resources. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021*, volume 207 of *LIPIcs*, pages 1:1–1:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.1.

13 Eric Budish. The Combinatorial Assignment Problem: Approximate Competitive Equilibrium from Equal Incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.

14 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum nash welfare. *ACM Transactions on Economics and Computation*, 7(3):12, 2019.

15 Hau Chan, Jing Chen, Bo Li, and Xiaowei Wu. Maximin-aware allocations of indivisible goods. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19*, pages 1871–1873, 2019. URL: http://dl.acm.org/citation.cfm?id=3331947.

16 Bhaskar Ray Chaudhury, Jugal Garg, and Kurt Mehlhorn. EFX exists for three agents. *J. ACM*, 71(1):4:1–4:27, 2024. doi:10.1145/3616009.

17 Bhaskar Ray Chaudhury, Telikepalli Kavitha, Kurt Mehlhorn, and Alkmini Sgouritsa. A little charity guarantees almost envy-freeness. *SIAM J. Comput.*, 50(4):1336–1358, 2021. doi:10.1137/20M1359134.

18 George Christodoulou, Amos Fiat, Elias Koutsoupias, and Alkmini Sgouritsa. Fair allocation in graphs. In Kevin Leyton-Brown, Jason D. Hartline, and Larry Samuelson, editors, *Proceedings of the 24th ACM Conference on Economics and Computation, EC 2023, London, United Kingdom, July 9-12, 2023*, pages 473–488. ACM, 2023. doi:10.1145/3580507.3597764.

19 Paul W. Goldberg, Kasper Høgh, and Alexandros Hollender. The frontier of intractability for EFX with two agents. In *Algorithmic Game Theory - 16th International Symposium, SAGT 2023, Egham, UK, September 4-7, 2023, Proceedings*, volume 14238 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2023. doi:10.1007/978-3-031-43254-5_17.

20 Hadi Hosseini, Sujoy Sikdar, Rohit Vaish, and Lirong Xia. Fair and efficient allocations under lexicographic preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5472–5480, 2021. doi:10.1609/AAAI.V35I6.16689.

21 Vishwa Prakash HV, Pratik Ghosal, Prajakta Nimbhorkar, and Nithin Varma. EFX exists for three types of agents. *CoRR*, abs/2410.13580, 2024. doi:10.48550/arXiv.2410.13580.

22 Alireza Kaviani, Masoud Seddighin, and AmirMohammad Shahrezaei. Almost envy-free allocation of indivisible goods: A tale of two valuations. In *Web and Internet Economics - 20th International Conference, WINE 2024, Proceedings*. Springer, 2024.

23 Richard J Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 125–131, 2004. doi:10.1145/988772.988792.

24 Ryoga Mahara. Existence of EFX for two additive valuations. *Discret. Appl. Math.*, 340:115–122, 2023. doi:10.1016/J.DAM.2023.06.035.

25 Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. *SIAM J. Discret. Math.*, 34(2):1039–1068, 2020. doi:10.1137/19M124397X.

26 Alkmini Sgouritsa and Minas Marios Sotiriou. On the existence of EFX allocations in multigraphs. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 2735–2737, 2025. doi:10.5555/3709347.3743995.

27 Alkmini Sgouritsa and Minas Marios Sotiriou. On the existence of EFX allocations in multigraphs. *CoRR*, abs/2502.09777, 2025. doi:10.48550/arXiv.2502.09777.

28 Jinghan A Zeng and Ruta Mehta. On the structure of EFX orientations on graphs. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, pages 2309–2316, 2025. doi:10.5555/3709347.3743871.

29 Yu Zhou, Tianze Wei, Minming Li, and Bo Li. A complete landscape of EFX allocations on graphs: goods, chores and mixed manna. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 3049–3056, 2024. URL: https://www.ijcai.org/proceedings/2024/338.

30 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690, 2006. doi:10.1145/1132516.1132612.

The Agafonov and Schnorr-Stimm Theorems for Probabilistic Automata

Laurent Bienvenu
LaBRI, CNRS & Université de Bordeaux, Talence, France

Hugo Gimbert
LaBRI, CNRS & Université de Bordeaux, Talence, France

Subin Pulari
LaBRI, CNRS & Université de Bordeaux, Talence, France

Abstract

For a fixed alphabet A, an infinite sequence X is said to be normal if every word w over A appears in X with the same frequency as any other word of the same length. A classical result of Agafonov (1966) relates normality to finite automata as follows: a sequence X is normal if and only if any subsequence of X selected by a finite automaton is itself normal. Another theorem of Schnorr and Stimm (1972) gives an alternative characterization: a sequence X is normal if and only if no gambler can win large amounts of money by betting on the sequence X using a strategy that can be described by a finite automaton. Both of these theorems are established in the setting of deterministic finite automata. This raises the question as to whether they can be extended to the setting of probabilistic finite automata. In the case of the Agafonov theorem, a partial positive answer was given by Léchine et al. (MFCS 2024) in a restricted case of probabilistic automata with rational transition probabilities.

In this paper, we settle the full conjecture by proving that both the Agafonov and the Schnorr-Stimm theorems hold true for arbitrary probabilistic automata. Specifically, we show that a sequence X is normal if and only if any probabilistic automaton selects a normal subsequence of X with probability 1 and also show that a sequence X is normal if and only if any probabilistic finite-state gambler fails to win on X with probability 1.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Normality, Agafonov theorem, probabilistic automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.16

Related Version *Extended Version*: https://arxiv.org/abs/2502.12307

Funding *Laurent Bienvenu*: supported in part by the ANR project FLITTLA ANR-21-CE48-0023.
Subin Pulari: supported in part by the ANR project FLITTLA ANR-21-CE48-0023.

1 Introduction

Given a finite alphabet A of k letters, an infinite sequence X of letters is said to be *normal* if every word w of A^* appear as subword of X with the same frequency as any other word of the same length, namely, $(1/k)^{|w|}$. The famous Champernowne sequence

$$0123456789101112131415161718192021222324252 6\ldots$$

can be shown to be normal over the alphabet $A = \{0, \ldots, 9\}$. The number π, for example, is conjectured to have a normal expansion in every base, though this very much remains an open question. Normal sequences are plentiful, and an easy way to generate a normal sequence X is to draw each letter $X(n)$ at random in the alphabet A (all letters having the same probability $1/|A|$) independently of the other chosen letters $X(m)$. The law of large numbers tells us that we obtain a normal sequence with probability 1.


© Laurent Bienvenu, Hugo Gimbert, and Subin Pulari;
licensed under Creative Commons License CC-BY 4.0



Of course, there are also plenty of examples of non-normal sequences:

- Periodic, or ultimately periodic, sequences can never be normal: indeed if the period of X is k there are only k possible subwords of X of length k hence most words of length k will not appear and their frequency will be 0.
- Sturmian sequences, which are sequences with only $k+1$ different subwords of length k (such as the Fibonacci sequence 1001010010010100101001001010010001... obtained by iterating the morphism $0 \mapsto 01$ and $1 \mapsto 0$) are not normal for the same reason.
- The Thue-Morse sequence 0110100110010110100101100110100 1... obtained by iterating the morphism $0 \mapsto 01$ and $1 \mapsto 10$ is not normal because 000 and 111 do not appear as subwords.
- A sequence of 0's and 1's generated at random where each bit is chosen equal to the previous one with probability 2/3 will have, with probability 1, all possible finite words as subwords but will not (still with probability 1) be normal as for example the word 00 will appear with frequency 1/3 instead of 1/4.

It turns out that normality has a nice interpretation in terms of finite automata. Indeed, two classical results, one due to Agafonov [1] and the other due to Schnorr and Stimm [9], assert that an infinite sequence is normal if and only if it cannot be predicted by a finite automaton with better-than-average accuracy. Of course, one needs to specify what "predicted" means. We consider two prediction models.

(I) In the Agafonov model, an automaton reads the infinite sequence X one letter at a time and updates its state in the usual way. Some of its states have a "select" tag on them. When the current state has such a tag, the next letter will be selected and added to a subsequence Y. We consider the automaton successful at predicting X if the subsequence Y built in this process is infinite and some letter of A does not have asymptotic frequency $1/|A|$ in Y. This means that the automaton has exhibited a statistical anomaly in the sequence X and isolated this anomaly in the subsequence Y.

(II) In the Schnorr-Stimm model, the predictor is still an automaton but this time is viewed as a gambling strategy. The gambler starts with a capital of \$1. Each state q is labeled with a betting function $\gamma_q : A \to \mathbb{R}^{\geq 0}$. This function represents the amount by which the predictor would like her capital to be multiplied by depending on the value of the next bit. For example, suppose the player plays against a sequence $X \in \{a, b, c\}^\omega$. If her current capital is \$2 and the current state q is labelled by a betting function γ_q such that $\gamma_q(a) = 0.7$, $\gamma_q(b) = 1.1$ and $\gamma(c) = 1.2$, if the next letter is a, her new capital will be \$1.4, if it is b her new capital will be \$2.2 and if it is c her new capital will be \$2.4. For the game to be fair, each betting function γ_q must have expectation equal to 1, i.e., must satisfy $\frac{1}{|A|}\sum_{a\in A}\gamma_q(a) = 1$. We say that the predictor wins if the capital of the player takes arbitrarily large values throughout the (infinite) game. That is, the predictor has spotted some type of statistical anomaly and is exploiting it to get rich!

Both of these models lead to the same conclusion: an infinite sequence X is normal if and only if it is unpredictable (by a finite automaton).

▶ **Theorem 1** (Agafonov [1]). *For $X \in A^\omega$, the following are equivalent.*

(i) *X is normal.*

(ii) *For any automaton $\mathcal{A}$ that selects a subsequence Y of X as in model I, either Y is finite, or every letter of A appears in Y with asymptotic frequency $1/|A|$.*

(iii) *For any automaton $\mathcal{A}$ that selects a subsequence Y of X as in model I, either Y is finite, or Y is normal.*

(See also Carton [3] and Seiller and Simonsen [10] for a modern account of the above theorem).

▶ **Theorem 2** (Schnorr-Stimm [9]). *For $X \in A^\omega$, the following are equivalent.*

(i) *X is normal.*

(ii) *Any automaton $\mathcal{A}$ betting on X according to model II does not win.*

In both of these theorems, the finite automata used for prediction are assumed to be deterministic. Would the situation change if one allowed probabilistic automata? In principle, one would not expect an unpredictable sequence to become predictable in the presence of a random source. Indeed, given a sequence X and a random source R it seems, informally speaking, that almost surely R will not "know" anything about X and thus will not help in predicting X. Surprisingly, this intuition is wrong in the setting where the predictors are not finite automata but are Turing machines, as shown by Bienvenu et al. [2] who built a sequence that is unpredictable by deterministic Turing machines (in either prediction model of selection or gambling) and becomes predictable (in either model) if one allows probabilistic Turing machines. Nonetheless, finite automata are much weaker than Turing machines and Bienvenu et al.'s construction does not work for such a memoryless model of computation. Indeed, recently, Léchine et al. showed that Agafonov's theorems holds for probabilistic automata in the restricted case where the transition probabilities are rational.

▶ **Theorem 3** (Léchine et al. [6]). *For $X \in A^\omega$, the following are equivalent.*

(i) *X is normal.*

(ii) *For any probabilistic automaton $\mathcal{A}$ with rational probabilities, almost surely, $\mathcal{A}$ selects a subsequence Y of X that is either finite, or every letter of A appears in Y with asymptotic frequency $1/|A|$.*

(iii) *For any probabilistic automaton $\mathcal{A}$ with rational probabilities, almost surely, $\mathcal{A}$ selects a subsequence Y of X such that either Y is finite, or Y is normal.*

This led them to conjecture that the probabilistic version of Agafonov's theorem holds in the general case, where the probabilities are real-valued rather than being rational. In this paper, we prove this conjecture and also prove the probabilistic version of the Schnorr-Stimm theorem. Additionally, we establish a probabilistic version of the Schnorr-Stimm dichotomy regarding the winning rates of probabilistic gamblers.

Léchine et al.'s proof is a reduction of the rational probabilistic case to the deterministic case by some clever combinatorial reduction (which substantially increases the number of states). Our proof is also a reduction to the deterministic case but is of very different nature: Instead of encoding a probabilistic automaton into a deterministic one, we will use the fact that a probabilistic automaton can be seen as a deterministic one over a larger alphabet, without altering the number of states. For this, we will need an extension of the deterministic case to Bernoulli measures (an extension which was proved by Seiller and Simonsen [10]), which will be presented in the next section.

Notation and terminology

We finish this introduction by formalizing the concepts discussed so far and gathering the notation and terminology that will be used in the rest of the paper.

Given an alphabet A, we denote by A^* the set of finite words over A, by A^n the set of words of length n, by A^ω the set of infinite sequences of letters and by $A^{\leq\omega}$ the set $A^* \cup A^\omega$. For $X \in A^{\leq\omega}$, we denote by $X(i)$ the i-th letter of X (by convention there is a 0-th letter) and by $X[i,j]$ the word $X(i)X(i+1)\dots X(j)$. Let ϵ denote the empty string.

Given a word u of length k and a word w of length $n \geq k$, we denote by $\mathrm{NbOcc}(u, w)$ the number of occurrences of the word u in w, i.e.,

$$\mathrm{NbOcc}(u, w) = \#\{i : 0 \leq i \leq n-k,\ w[i, i+k-1] = u\}$$

and the frequency of occurrence $\mathrm{Freq}(u, w)$ of u in w is naturally defined by

$$\mathrm{Freq}(u, w) = \frac{\mathrm{NbOcc}(u, w)}{n-k+1}$$

When X is an infinite sequence, we define

$$\mathrm{Freq}^-(u, X) = \liminf_{n\to\infty} \mathrm{Freq}(u, X[0, n]) \quad \text{and} \quad \mathrm{Freq}^+(u, X) = \limsup_{n\to\infty} \mathrm{Freq}(u, X[0, n])$$

When $\mathrm{Freq}^-(u, X)$ and $\mathrm{Freq}^+(u, X)$ have the same value, we simply call this common value $\mathrm{Freq}(u, X)$.

Given $X \in A^\omega$, we say that X is *balanced* if all letters appear in X with the expected frequency, i.e., $\mathrm{Freq}(a, X) = 1/|A|$ for all $a \in A$. We say that X is *normal* if all words appear in X with the expected frequency, i.e., $\mathrm{Freq}(w, X) = |A|^{-|w|}$ for all $w \in A^*$.

A *deterministic finite automaton (DFA)* is a tuple (Q, A, q_I, δ) where Q is a finite set of states, A a finite alphabet, q_I the initial state and $\delta : Q \times A \to Q$ the transition function (in this paper, runs of automata are meant to be infinite hence there is no need for final states). We denote by δ^* the function from $Q \times A^*$ to Q defined inductively by $\delta^*(q, \epsilon) = q$ and for $w \in A^*$ and $a \in A$, $\delta^*(q, w \cdot a) = \delta(\delta^*(q, w), a)$, where $\cdot$ is the concatenation of words.

An *automatic selector* (or *selector* for short) is a tuple (Q, A, q_I, δ, S) where (Q, A, q_I, δ) is a DFA and S is a subset of Q, representing the selection states.

Given a selector $\mathcal{S} = (Q, A, q_I, \delta, S)$, we define the selection function from A^* to A^* inductively by:

$$\mathrm{Select}(\mathcal{S}, \epsilon) = \epsilon$$

and for $w \in A^*$ and $a \in A$:

$$\mathrm{Select}(\mathcal{S}, w \cdot a) = \begin{cases} \mathrm{Select}(\mathcal{S}, w) \cdot a & \text{if } \delta^*(q_I, w) \in S \\ \mathrm{Select}(\mathcal{S}, w) & \text{if } \delta^*(q_I, w) \notin S \end{cases}$$

If X is an infinite sequence in A^ω, the sequence of words $\mathrm{Select}(\mathcal{S}, X[0, n])$ is non-decreasing with respect to the prefix order and thus converges to a sequence $Y \in A^{\leq\omega}$ which we call *the selected subsequence of X selected by $\mathcal{S}$* and denote by $\mathrm{Select}(\mathcal{S}, X)$.

An *automatic gambler* (or *gambler* for short) is a tuple $(Q, A, q_I, \delta, \gamma)$ where (Q, A, q_I, δ) is a DFA and γ is a function from $Q \times A$ to $\mathbb{R}^{\geq 0}$ such that for all q, $\frac{1}{|A|}\sum_{a\in A} \gamma(q, a) = 1$. As said earlier, the value of $\gamma(q, a)$ should be interpreted as the multiplier by which the gambler, being currently in state q, would like her capital to be multiplied by if the next read letter is a. The condition $\frac{1}{|A|}\sum_{a\in A} \gamma(q, a) = 1$ ensures that the game is fair.

Given a gambler $\mathcal{G} = (Q, A, q_I, \delta, \gamma)$ and $w \in A^*$, we define $\mathrm{Capital}(\mathcal{G}, w)$ inductively by

$$\mathrm{Capital}(\mathcal{G}, \epsilon) = 1$$

and for $w \in A^*$ and $a \in A$,

$$\mathrm{Capital}(\mathcal{G}, w \cdot a) = \mathrm{Capital}(\mathcal{G}, w) \cdot \gamma(\delta^*(q_I, w), a)$$

and we say that a gambler $\mathcal{G}$ *wins against* $X \in A^\omega$ if

$$\limsup_{n\to+\infty} \mathrm{Capital}(\mathcal{G}, X[0, n]) = +\infty$$

(otherwise we say that $\mathcal{G}$ loses).

2 Deterministic prediction for Bernoulli measures

In classical normality, all letters of the alphabet occur with the same frequency. We can however consider the extension of normality to Bernoulli measures. A Bernoulli measure over A^ω is a probability measure where letters of an infinite sequence X are drawn at random independently of one another but the distribution over the alphabet A is non-uniform.

▶ **Definition 4.** *Let $\mu : A \to [0,1]$ be a distribution over the alphabet A (hence satisfying $\sum_{a\in A} \mu(a) = 1$). The Bernoulli measure induced by μ, which we also denote by μ by abuse of notation, is the unique probability measure such that for all i, k, for every word $w = a_0, \dots a_k \in A$,*

$$\Pr_{X\sim\mu}\Big[X[i, i+k] = w\Big] = \prod_{j=0}^{k} \mu(a_j)$$

We also denote by $\mu(w)$ the quantity $\prod_{j=0}^{k} \mu(a_j)$.

Normality generalizes very naturally to Bernoulli measures.

▶ **Definition 5.** *Let μ be a Bernoulli measure. A sequence $X \in A^\omega$ is* μ-balanced *if* $\mathrm{Freq}(a, X) = \mu(a)$ *for all $a \in A$. It is* μ-normal *if for all words $w \in A^*$, $\mathrm{Freq}(w, X) = \mu(w)$.*

We say that a Bernoulli measure μ is *positive* when $\mu(a) > 0$ for every letter a. In the rest of the paper, all Bernoulli measures will be assumed to be positive, and we simply say "Bernoulli measure" to mean "positive Bernoulli measure".

The Agafonov theorem can be extended to Bernoulli measures, as proven by Seiller and Simonsen [10]. It is this theorem that we will use in the next section to obtain a proof of the Agafonov theorem for probabilistic selectors.

▶ **Theorem 6** (Agafonov theorem for Bernoulli measures [10]). *For $X \in A^\omega$, the following are equivalent.*

- **(i)** *X is μ-normal.*
- **(ii)** *For any selector $\mathcal{S}$ that selects a subsequence Y of X, either Y is finite or Y is μ-balanced.*
- **(iii)** *For any selector $\mathcal{S}$ that selects a subsequence Y of X, either Y is finite or Y is μ-normal.*

We can also easily generalize the notion of gambler to the setting of Bernoulli measures: it suffices to define a μ-gambler $\mathcal{G} = (Q, A, q_I, \delta, \gamma)$ as before but with the fairness condition on γ replaced by $\sum_{a\in A} \mu(a)\gamma(q, a) = 1$ for every q. The function Capital and the notion of success are defined as before.

We will now prove that the Schnorr-Stimm theorem, just like the Agafonov theorem, can also be extended to Bernoulli measures.

▶ **Theorem 7** (Schnorr-Stimm theorem for Bernoulli measures). *For $X \in A^\omega$ and a Bernoulli measure μ, the following are equivalent.*

- **(i)** *X is μ-normal.*
- **(ii)** *No μ-gambler $\mathcal{G}$ wins by betting on X.*

Proof.

$(i) \Rightarrow (ii)$. Suppose that $X \in A^\omega$ is μ-normal and consider a μ-gambler $\mathcal{G} = (Q, A, q_I, \delta, \gamma)$.

We can assume that the μ-gambler only has one state on which it places a non-trivial bet. Indeed, define for every $q \in Q$ the μ-gambler $\mathcal{G}^{[q]} = (Q, A, q_I, \delta, \gamma^{[q]})$ where $\gamma^{[q]}(q, a) = \gamma(q, a)$ for all a and $\gamma^{[q]}(q', a) = 1$ for $q' \neq q$. That is, $\mathcal{G}^{[q]}$ is the gambler $\mathcal{G}$ where all states but state q are neutralized (no bet is placed while on them). By the multiplicative nature of the function Capital, we have for all n:

$$\mathrm{Capital}(\mathcal{G}, X[0,n]) = \prod_{q \in Q} \mathrm{Capital}(\mathcal{G}^{[q]}, X[0,n])$$

Thus if we can show that all quantities $\mathrm{Capital}(\mathcal{G}^{[q]}, X[0,n])$ are bounded, we are done. Let us assume that there is a state r that is the unique state on which $\mathcal{G}$ bets. If instead of $\mathcal{G}$ we consider the selector $\mathcal{S} = (Q, A, q_I, \delta, \{r\})$ with only r as the selecting state, we know by Agafonov's theorem for Bernoulli measures (Theorem 6) that the subsequence Y of X selected by $\mathcal{S}$ is μ-normal, hence in particular μ-balanced. But this subsequence is precisely the values of X on which $\mathcal{G}$ bets!

We can further assume that Y is infinite, otherwise it means that the run of $\mathcal{G}$ on X passes by r finitely often, hence $\mathcal{G}$ certainly cannot win as other states are not betting states. Now, suppose that at the n-th step of the run on X the state r has been visited $k = k(n)$ times. We have

$$\mathrm{Capital}(\mathcal{G}, X[0,n]) = \prod_{a \in A} \gamma(r,a)^{\mathrm{NbOcc}(a, Y[0,k])}$$

But since Y is μ-normal we have, for every a, $\mathrm{NbOcc}(a, Y[0,k]) = \mu(a)k + o(k)$. Thus,

$$\mathrm{Capital}(\mathcal{G}, X[0,n]) = \prod_{a \in A} \gamma(r,a)^{\mu(a)k + o(k)}$$

or equivalently

$$\log \mathrm{Capital}(\mathcal{G}, X[0,n]) = (k + o(k)) \cdot \sum_{a \in A} \mu(a) \cdot \log \gamma(r,a)$$

(here we assume that all values $\gamma(r,a)$ involved in the product are positive for if not then the capital falls to 0 and we are done). Since we have $\sum_{a \in A} \mu(a) = 1$ (μ being a distribution), we can use the strict concavity of the function log on $(0, +\infty)$ to apply Jensen's inequality and get

$$\sum_{a \in A} \mu(a) \cdot \log \gamma(r,a) \leq \log \left(\sum_{a \in A} \mu(a) \gamma(r,a) \right)$$

with strict inequality when not all $\gamma(q,a)$ are equal (which is the case where $\mathcal{G}$ makes non-trivial bets). But by the fairness condition, we have $\sum_{a \in A} \mu(a)\gamma(r,a) = 1$ hence we see that $\log \mathrm{Capital}(\mathcal{G}, X[0,n])$ is either 0 or ultimately negative which either way means that $\mathcal{G}$ does not win.

$(ii) \Rightarrow (i)$. Assume that X is not μ-normal. This means that there is some word w such that $\mathrm{Freq}(w, X[0,n])$ does not converge to $\mu(w)$. Let us assume that w is such a word of minimal length and write $w = ux$ with $u \in A^*$ and $x \in A$.

Consider the sequence of vectors f_n defined by

$$f_n = \left(\frac{\mathrm{Freq}(ua, X[0,n])}{\sum_{b\in A} \mathrm{Freq}(ub, X[0,n])} \right)_{a\in A}$$

or equivalently,

$$f_n = \left(\frac{\mathrm{NbOcc}(ua, X[0,n])}{\sum_{b\in A} \mathrm{NbOcc}(ub, X[0,n])} \right)_{a\in A}$$

All of these vectors belong to the set $\Gamma = \{f : A \to [0,1] : \sum_{a\in A} f(a) = 1\}$. This is a compact set, hence the sequence f_n must have cluster points. By definition of u and x, we know that f_n does not converge to μ because $f_n(x)$ does not converge to $\mu(x)$: Indeed, in the definition of $f_n(x)$, the denominator converges to $\mathrm{Freq}(u, X)$ which by minimality of w is defined and equal to $\mu(u)$, while the numerator is equal to $\mathrm{Freq}(w, X[0,n])$ which by definition of w does not converge to $\mu(w) = \mu(u)\mu(x)$.

Therefore, the sequence (f_n) must have at least one cluster point ν different from μ. Fix such a cluster point ν.

We now build our gambler $\mathcal{G} = (Q, A, q_I, \delta, \gamma)$. The idea is that the gambler will record the last $|u|$ bits it read and will only place bets when these exactly form the word u. Let us thus take $Q = \{q_v : v \in A^*, |v| \leq |u|\}$ initial state $q_I = q_\epsilon$ and define δ by

$$\delta(q_v, a) = \begin{cases} q_{va}, & \text{if } |v| < |u|, \\ q_{v'a} & \text{if } |v| = |u| \text{ and } v = xv' \text{ with } x \in A. \end{cases}$$

Now, define $\gamma(q_v, a) = 1$ whenever $v \neq u$ and $\gamma(q_u, a) = \nu(a)/\mu(a)$ for all $a \in A$. Observe that this is a valid μ-gambler as the fairness condition is satisfied: $\sum_{a\in A} \nu(a)/\mu(a) \cdot \mu(a) = \sum_a \nu(a) = 1$.

Suppose that after reading n letters of X the state q_u has been visited $k = k(n)$ times. First, observe that $k(n)$ tends to $+\infty$. Indeed, the state q_u is visited whenever u is seen as a subword of X. But we assumed that u appears in X with frequency $\mu(u)$, by minimality of w.

Second, unfolding the definition of f_n, we have,

$$\mathrm{Capital}(\mathcal{G}, X[0,n]) = \prod_{a\in A} \left(\frac{\nu(a)}{\mu(a)} \right)^{k\cdot f_n(a)}.$$

This gives

$$\log \mathrm{Capital}(\mathcal{G}, X[0,n]) = k \cdot \sum_{a\in A} f_n(a) \log \left(\frac{\nu(a)}{\mu(a)} \right).$$

Since ν is a cluster point of the sequence f_n, for any fixed $\varepsilon > 0$ there are infinitely many n (or k) such that

$$\sum_{a\in A} f_n(a) \log \left(\frac{\nu(a)}{\mu(a)} \right) \geq \sum_{a\in A} \nu(a) \log \left(\frac{\nu(a)}{\mu(a)} \right) - \varepsilon.$$

But the term $\sum_{a\in A} \nu(a) \log \left(\frac{\nu(a)}{\mu(a)} \right)$ is the relative entropy from μ to ν (also known as Kullback-Liebler divergence, see for example [5]) which we denote by $\mathrm{D}_{KL}(\nu||\mu)$. This

quantity is non-negative in general and is positive when $\nu \neq \mu$, which is the case here. We have thus established that for any fixed ε, there are arbitrarily large n and k such that

$$\log \mathrm{Capital}(\mathcal{G}, X[0,n]) \geq k\left(\mathrm{D}_{KL}(\nu||\mu) - \varepsilon\right).$$

Taking any $\varepsilon < \mathrm{D}_{KL}(\nu||\mu)$, this shows that

$$\limsup_{n\to+\infty} \mathrm{Capital}(\mathcal{G}, X[0,n]) = +\infty. \qquad \blacktriangleleft$$

Let us note that this last proof actually gives us a finer analysis of normality, in terms of the rate of failure or success of the gambler. This was already observed by Schnorr and Stimm in their seminal paper, where they proved the following.

▶ **Theorem 8** (Schnorr-Stimm dichotomy theorem [9]). *Let X be an infinite sequence in A^ω.*

(i) *If X is normal and $\mathcal{G}$ is a gambler, then the capital of $\mathcal{G}$ throughout the game either is ultimately constant or decreases at an exponential rate.*

(ii) *If X is not normal, then there exists a gambler $\mathcal{G}$ which wins against X at an "infinitely often" exponential rate (i.e., $\limsup_n \log(\mathrm{Capital})/n > 0$).*

As a byproduct of our proof of Theorem 7, we have the same dichotomy for positive Bernoulli measures (i.e., Bernoulli measures such that $\mu(a) > 0$ for every letter):

▶ **Theorem 9** (Schnorr-Stimm dichotomy theorem for Bernoulli measures). *Let X be an infinite sequence in A^ω and μ a positive Bernoulli measure.*

(i) *If X is μ-normal and $\mathcal{G}$ is a μ-gambler, then the capital of $\mathcal{G}$ throughout the game either is ultimately constant or decreases at an exponential rate.*

(ii) *If X is not μ-normal, then there exists a μ-gambler $\mathcal{G}$ which wins against X at an "infinitely often" exponential rate.*

Our proof of Theorem 7 almost establishes this, but we do need an additional technical lemma and the block-wise characterization of μ-normality. Let $\mathrm{BFreq}(u, w)$ denote the block frequency of the word u in w, defined as the proportion of non-overlapping blocks in w that are equal to u. More precisely when $n = |w|$ and $k = |u|$,

$$\mathrm{BFreq}(u, w) = \frac{\#\{i : 0 \leq i \leq \lfloor n/k \rfloor \ ,\ w[ki, k(i+1)-1] = u\}}{\lfloor n/k \rfloor}.$$

For $w \in A^*$, let $\mathrm{BFreq}^-(w, X)$, $\mathrm{BFreq}^+(w, X)$ and $\mathrm{BFreq}(w, X)$ denote the lower, upper and limit block frequency of w in X defined similarly as $\mathrm{Freq}^-(w, X)$, $\mathrm{Freq}^+(w, X)$ and $\mathrm{Freq}(w, X)$. A sequence X is *μ-block normal* if for all words $w \in A^*$, $\mathrm{BFreq}(w, X) = \mu(w)$. The following was shown by Seiller and Simonsen.

▶ **Lemma 10** (Seiller-Simonsen [10]). *If a sequence $X \in A^\omega$ is μ-normal then X is μ-block normal.*

We note that the converse implication also holds.

▶ **Lemma 11.** *If a sequence $X \in A^\omega$ is μ-block normal then X is μ-normal.*

The above lemmas completes the proof of the following equivalence theorem between μ-normality and μ-block normality.

▶ **Theorem 12.** *A sequence $X \in A^\omega$ is μ-normal if and only if X is μ-block normal.*

The proof of Lemma 11 uses a counting trick from the proof of Theorem 3.1 from [8] which in turn is based on the proof of the main theorem in [7].

Proof of Lemma 11. As in the proof of Theorem 3.1 from [8], for any finite length string $w = a_1a_2a_3\dots a_k \in A^k$ and large enough n,

$$\mathrm{Freq}(w, X[0,n]) = f_1(n) + f_2(n) + \cdots + f_{(1+\lfloor \log_2 \frac{n}{k} \rfloor)}(n) + \frac{(k-1)\cdot O(\log n)}{n-k+1} \tag{1}$$

where $f_p(n)$ are defined as follows:

$$f_p(n) = \begin{cases} \frac{|\{i \mid X[ki,k(i+1)-1]=w \ , \ 0 \le i \le \lfloor n/k \rfloor\}|}{n-k+1}, & \text{if } \ p=1 \\ \sum_{j=1}^{k-1} \frac{|\{i | X[2^{p-1}ki, 2^{p-1}k(i+1)-1] \in S_j, 0 \le i \le n/2^{p-1}k\}|}{n-k+1}, & \text{if } 1 < p \le (1+\lfloor \log_2(n/k) \rfloor) \\ 0, \ \text{otherwise.} \end{cases}$$

In the above definition, S_j is the set of strings of the form, $u\ a_1a_2\dots a_k\ v$ where u is some string of length $2^{p-2}k - j$ and v is some string of length $2^{p-2}k - k + j$.

Equation 1 shows that the frequency of w in $X[0,n]$ can be written as a sum of different block frequencies. The quantity $f_1(n)$ counts the number of occurrences of w inside disjoint k-length blocks in $X[0,n]$, $f_2(n)$ counts the number of occurrences crossing the boundaries of these k-blocks, $f_3(n)$ counts those crossing boundaries of $2k$-blocks, and so on. In general, $f_p(n)$ counts the number of occurrences straddling boundaries of disjoint $2^{p-1}k$-length blocks. Each $f_p(n)$ may miss at most $(k-1)$ occurrences of w, which are accounted for by the error term.

Since X is μ-block normal,

$$\lim_{n\to\infty} f_1(n) = \lim_{n\to\infty} \frac{|\{i \mid X[ki,k(i+1)-1] = w \ , \ 0 \le i \le \lfloor n/k \rfloor\}|}{n-k+1} = \frac{\mu(w)}{k}.$$

Now, when $p \le 1 + \left\lfloor \log_2(\frac{n}{k}) \right\rfloor$,

$$\begin{aligned} \lim_{n\to\infty} f_p(n) &= \sum_{j=1}^{k-1} \lim_{n\to\infty} \frac{|\{i \mid X[2^{p-1}ki, 2^{p-1}k(i+1)-1] \in S_j, 0 \le i \le n/2^{p-1}k\}|}{n-k+1} \\ &= \frac{\mu(w)}{2^{p-1}k}(k-1). \end{aligned}$$

Since $\langle \sum_{i=1}^{m} f_i(n) \rangle_{m\in\mathbb{N}}$ is uniformly convergent, we have

$$\mathrm{Freq}(w,X) = \lim_{n\to\infty} \sum_{i=1}^{\infty} f_i(n) = \sum_{i=1}^{\infty} \lim_{n\to\infty} f_i(n).$$

Therefore from (1),

$$\lim_{n\to\infty} \mathrm{Freq}(w, X[0,n]) = \frac{\mu(w)}{k} + (k-1)\mu(w)\Big[\sum_{i=1}^{\infty} \frac{1}{2^i k}\Big] = \mu(w).$$

Hence, X is μ-normal. ◀

We require the following technical lemma in the proof of Theorem 9.

▶ **Lemma 13.** *Let (Q, A, q_I, δ) be a finite-state automaton, μ be a positive Bernoulli measure and let $V_q(n, X)$ denote the number of times the state q is visited upon running the automaton using the first n bits of $X \in A^\omega$.*

Then, for every $q \in Q$, there exists a real number $\pi_q \ge 0$ such that, for every μ-normal sequence $X \in A^\omega$:

- *either* $V_q(n, X)$ *is ultimately constant (i.e., q is visited only finitely often during the run on X)*
- *or,* $\lim_{n\to\infty} \frac{V_q(n,X)}{n} = \pi_q$ *(i.e., the state q is visited with asymptotic frequency π_q) and this second case can only happen when $\pi_q > 0$.*

Proof. On running an automaton upon a normal sequence, starting from any state, a strongly connected component must be reached in finitely many steps. Similar to [10], let us consider the Markov chain corresponding to the $|Q| \times |Q|$ stochastic matrix $\mathbf{P}$ where,

$$\mathbf{P}_{ij} = \sum_{a\in A} \mu(a) \cdot 1_{\delta(i,a)=j}.$$

The proof follows using the Ergodic Theorem for Markov chains and the same steps in the proof of Lemma 4.5 from [9] by replacing the uniform measure with the positive Bernoulli measure induced by μ. We note that this line of proof uses μ-block normality, which is equivalent to μ-normality (Theorem 12). ◀

Proof of Theorem 9. In part $(i) \Rightarrow (ii)$ of our proof of Theorem 7, we showed that on a given state r, either r is not a betting state (γ is the constant 1 on this state) or it is and then the gambler loses money exponentially fast in the number of times this state is visited, the exponent being $\alpha_r = \sum_{a\in A} \mu(a) \cdot \log \gamma(r, a)$ which we proved to be negative. By Lemma 13, betting states are either visited finitely often or with positive asymptotic density. If they are all visited finitely often, the capital stabilizes after the last bet is made. Otherwise, if betting states r are visited with frequency π_r and at least one π_r is positive, then the gambler loses at an exponential rate, where the exponent is $\sum_r \pi_r \alpha_r$.

In part $(ii) \Rightarrow (i)$, under the assumption of non-μ-normality of X, we built a gambler $\mathcal{G}$ satisfying the following: for any fixed ε, there are arbitrarily large n and k such that $\log \text{Capital}(\mathcal{G}, X[0, n]) \geq k\,(\text{D}_{KL}(\nu||\mu) - \varepsilon)$. Here, k is the number of visits to a state q_u. In the proof of Theorem 7 part (ii), this state q_u is visited with frequency $\mu(u)$ for large enough n. Hence the gambler has an "infinitely often" exponential rate of success with exponent $\mu(u)D_{KL}(\nu||\mu)$. ◀

3 The Agafonov theorem for PFAs

We now want to prove the extension of Theorem 6 to probabilistic automata/selectors. A *probabilistic finite automaton (PFA)* is a tuple (Q, A, q_I, δ) where Q is a finite set of states, A is a finite alphabet, q_I is the initial state and $\delta : Q \times A \to \Delta(Q)$ is a probabilistic transition function, that is, $\Delta(Q)$ is the set of probability distributions over Q. In this setting, we define inductively the random variables $\delta^*(q, w)$ by $\delta^*(q, \epsilon) = q$ and for $w \in A^*$ and $a \in A$, the event $\delta^*(q, w \cdot a) = q'$ is defined as the union

$$\bigcup_{r\in Q} \left[\delta^*(q, w) = r \wedge \delta_{|w|+1}(r, a) = q'\right]$$

where $\{\delta_n(r, b) : n \in \mathbb{N}, r \in Q, b \in A\}$ is a family of independent random variables such that for all (n, r, b), the distribution of $\delta_n(r, b)$ is $\delta(r, b)$.

Modulo this change of type of transition, probabilistic selectors as well as Select are defined as before. This makes $\text{Select}(\mathcal{S}, X)$ a random variable for every given $X \in A^{\leq\omega}$.

In order to lift the deterministic Agafonov theorem for Bernoulli measures to the probabilistic case, we will need some preliminary lemmas about normality.

Given two alphabets A and B, and given two sequences $X \in A^\omega$ and $Y \in B^\omega$, we denote by $X \otimes Y$ the sequence Z over the alphabet $A \times B$ where $Z(n) = (X(n), Y(n))$. For two words v and w of the same length over A and B respectively, the product $v \otimes w$ is defined in the same way. Likewise, if μ and ν are Bernoulli measures over A and B respectively, $\mu \otimes \nu$ is the Bernoulli measure ξ over $A \times B$ where $\xi((a,b)) = \mu(a)\nu(b)$, for all $(a,b) \in A \times B$. Finally, if Z is a sequence over a product alphabet $A \times B$, we denote by π_0 and π_1 its first and second projection respectively (in other words, $\pi_0(X \otimes Y) = X$ and $\pi_1(X \otimes Y) = Y$ for all X, Y).

▶ **Lemma 14.** *Let A and B be two alphabets and μ, ν two Bernoulli measures over A and B respectively. If a sequence $X \in A^\omega$ is μ-normal and a sequence $Y \in B^\omega$ is drawn at random according to ν, then ν-almost surely*[1]*, $X \otimes Y$ is $\mu \otimes \nu$-normal.*

Proof. While it is a bit easier to prove this lemma for block-normality, we prefer a more direct proof which does not rely on the equivalence between normality and block-normality.

Let $w = u \otimes v$ be a non-empty word over $(A \times B)^*$. Let N be a large integer. We split $X \otimes Y$ into blocks of length N :

$$X \otimes Y = (X_1 \otimes Y_1) \cdot (X_2 \otimes Y_2) \cdot \dots$$

with $|X_i| = |Y_i| = N$, for all $i \geq 1$. Introduce the random variables $(B_i)_{i \geq 1}$ which count the number of occurrences of $u \otimes v$ in each block:

$$B_i = \mathrm{NbOcc}(u \otimes v, X_i \otimes Y_i) \ .$$

For every integer $n \in \mathbb{N}$, within $(X \otimes Y)[0, n]$ there are $\lfloor n/N \rfloor$ complete blocks. Some occurrences of $u \otimes v$ in $(X \otimes Y)[0, n]$ do occur inside a block $X_i \otimes Y_i$ while some other do not because they overlap two contiguous blocks. There can be at most $|w|$ such overlapping occurrences between two given blocks. That observation leads to two ways to count the number of occurrences of $u \otimes v$: the exact way C_n and the N-block way $C'_{n,N}$, two random variables satisfying:

$$\begin{aligned} C_n &= \mathrm{NbOcc}(u \otimes v, (X \otimes Y)[0, n]) \\ C'_{n,N} &= \sum_{i \in 1 \dots \lfloor n/N \rfloor} B_i \\ C'_{n,N} &\leq C_n \leq C'_{n+N,N} + |w| \cdot \lfloor n/N \rfloor \ . \end{aligned} \tag{2}$$

Let us focus on $C'_{n,N}$ first. The variables B_i can be grouped into $|A|^N$ different buckets, with respect to the corresponding value of X_i. For every $x \in A^N$, set $I_n(x) = \{1 \leq i \leq \lfloor n/N \rfloor \mid X_i = x\}$, which is non-empty for n large enough, since X is normal. The random variables $(X_i \otimes Y_i)$ are mutually independent, and so are the random variables $(B_i)_{i \geq 1}$. Moreover, for a fixed $x \in A^N$, the random variables $(B_i)_{i \in I_\infty(x)}$ are distributed identically, denote by $\xi_N(x)$ the corresponding probability distribution on $0 \dots N - |w| + 1$. Then $\mathbb{E}[\xi_N(x)]$ measures the expected number of occurrences of $u \otimes v$ in a sequence where the left part is fixed, equal to x, and the right part is independently generated according to ν. Thus

$$\mathbb{E}[\xi_N(x)] = \mathrm{NbOcc}(u, x) \cdot \nu(v) \ .$$

[1] For a probability 1 set of Y according to the measure ν.

According to the Law of Large Numbers, $\frac{1}{|I_n(x)|}\sum_{i\in I_n(x)} B_i \to_n \mathrm{NbOcc}(u,x)\cdot\nu(v)$. Since X is normal, every word $x\in A^N$ occurs with frequency $\mu(x)$ in $X_1, X_2, \ldots$ thus, almost-surely,

$$\lim_n \frac{1}{n}C'_{n,N} = \left(\sum_{x\in A^N} \mu(x)\cdot \mathrm{NbOcc}(u,x)\right)\cdot\nu(v) \ . \tag{3}$$

Since X is normal, the right part in (3) converges to $\mu(u)\cdot\nu(v)$ when N grows large. Using (2), we get the desired result:

$$\frac{1}{n}C_n \to_n \lim_N \left(\lim_n C'_{n,N}\right) = (\mu\otimes\nu)(u\otimes v) \ . \qquad \blacktriangleleft$$

▶ **Lemma 15.** *If a sequence Z is $\mu\otimes\nu$-normal over $A\times B$, then $\pi_0(Z)$ and $\pi_1(Z)$ are μ-normal and ν-normal respectively.*

Proof. Suppose $Z\in(A\times B)^\omega$ is $\mu\otimes\nu$-normal. We only need to show that $\pi_0(Z)$ is μ-normal, the proof of the ν-normality of $\pi_1(Z)$ is the same by symmetry. Let $w\in A^n$. We have

$$\mathrm{Freq}^+(w,\pi_0(Z)) = \sum_{w'\in B^n} \mathrm{Freq}^+(w\otimes w', Z)$$

which, by $\mu\otimes\nu$-normality of Z, implies

$$\mathrm{Freq}^+(w,\pi_0(Z)) = \sum_{w'\in B^n} \mu\otimes\nu(w\otimes w') = \sum_{w'\in B^n}\mu(w)\nu(w') = \mu(w).$$

The same holds for $\mathrm{Freq}^-(w,\pi_0(Z))$, hence we have proven that $\mathrm{Freq}(w,\pi_0(Z)) = \mu(w)$, which is what we wanted. $\blacktriangleleft$

We are now ready to prove Agafonov's theorem for PFAs.

▶ **Theorem 16** (Agafonov's theorem for PFA)**.** *Let $X\in A^\omega$, and μ a Bernoulli measure over A. The following are equivalent.*

- **(i)** *X is μ-normal.*
- **(ii)** *For any deterministic selector $\mathcal{S}$ that selects a subsequence Y of X, either Y is finite or Y is μ-normal.*
- **(iii)** *For any probabilistic selector $\mathcal{S}$ that selects a subsequence Y of X, almost surely, either Y is finite, or Y is μ-normal.*

Proof. Since deterministic selectors (for which we already have Agafonov's theorem) are a subset of the probabilistic ones, all that is left to prove is $(i)\Rightarrow(iii)$.

Let $\mathcal{S} = (Q, A, q_I, \delta, S)$ be a probabilistic selector. Recall that each transition $\delta(q,a)$ (where $q\in Q$ and $a\in A$) is some probability distribution over Q. Consider the set $\mathcal{T}$ of all functions $Q\times A\to Q$. We can put a distribution τ over $\mathcal{T}$ such that if t is chosen according to τ, for every (q,a), the marginal distribution of $t(q,a)$ is $\delta(q,a)$. An easy way to do this is to take $\tau = \bigotimes_{(q,a)\in Q\times A}\delta(q,a)$.

This construction means that, for a fixed sequence $X\in A^\omega$, an equivalent way to simulate the probabilistic run of $\mathcal{S}$ on X is, every time we are in a state q and read a letter a, to pick t at random according to τ and move to state $t(q,a)$. But $\mathcal{T}$ is a finite set and τ a Bernoulli measure over it. This Bernoulli measure might not be positive. If it is not, we simply remove from $\mathcal{T}$ all functions whose τ-probability is 0. Reformulating slightly, yet another equivalent way to simulate the run of $\mathcal{S}$ over X is to do the following:

1. First choose $T\in\mathcal{T}^\omega$ at random with respect to the Bernoulli measure τ.
2. Then run on the sequence $X\otimes T$ the deterministic selector $\hat{\mathcal{S}}$ whose set of states is Q and transition $\hat\delta$ is defined by $\hat\delta(q,(a,t)) = t(q,a)$.

Now, the two following random variables have the same distribution:

- The subsequence Y of X selected by $\mathcal{S}$.
- The sequence $\pi_0(\hat{Y})$, where $\hat{Y}$ is the subsequence of $X \otimes T$ selected by $\hat{\mathcal{S}}$ when T is chosen randomly according to τ.

Since X is μ-normal, by Lemma 14, $X \otimes T$ is $\mu \otimes \tau$-normal τ-almost surely. Thus, by Agafonov's theorem for deterministic selectors and Bernoulli measures (Theorem 6), almost surely, the subsequence $\hat{Y}$ selected by $\hat{\mathcal{S}}$ is $\mu \otimes \tau$-normal. Finally, by Lemma 15, this implies that almost surely, the subsequence $\pi_0(\hat{Y})$ of X is μ-normal. This concludes the proof. ◀

4 The Schnorr-Stimm theorem for probabilistic gamblers

Finally, we establish the Schnorr-Stimm theorem for probabilistic finite-state automata. The definition of probabilistic (μ-)gambler is the same as the definition of deterministic (μ-)gambler except that it is based on PFAs instead of DFAs. In this setting, the quantities $\mathrm{Capital}(\mathcal{G}, w)$ become random variables.

▶ **Theorem 17** (Schnorr-Stimm theorem for PFAs). *For $X \in A^\omega$ and μ a Bernoulli measure, the following are equivalent.*

(i) *X is μ-normal.*

(ii) *Any probabilistic μ-gambler $\mathcal{G}$ betting on X loses almost surely.*

Proof. We already have $(ii) \Rightarrow (i)$ by Theorem 7 (deterministic gamblers being a subset of probabilistic ones).

For $(i) \Rightarrow (ii)$, the idea is similar to the proof of Theorem 16. To simulate the run of a probabilistic μ-gambler $\mathcal{G} = (Q, A, q_I, \delta, \gamma)$ on a sequence X, one can equivalently run on the sequence $X \otimes T$ (where $T \in \mathcal{T}^\omega$ is τ-random sequence)[2] the deterministic gambler $\hat{\mathcal{G}} = (Q, A, q_I, \hat{\delta}, \hat{\gamma})$ where again $\hat{\delta}(q, (a, t)) = t(q, a)$ for all $(q, a, t) \in Q \times A \times \mathcal{T}$ and $\hat{\gamma}(q, (a, t)) = \gamma(q, a)$ (indeed the bet placed by a gambler at a given stage does not take into account which state will be reached next). Note that the fairness condition is respected since for every q,

$$\begin{aligned}
\sum_{(a,t)\in A\times\mathcal{T}} (\mu\otimes\tau)(a,t)\cdot\hat{\gamma}(q,(a,t)) &= \sum_{a\in A, t\in\mathcal{T}} \mu(a)\tau(t)\gamma(q,a) \\
&= \sum_{t\in\mathcal{T}} \tau(t) \sum_{a\in A} \mu(a)\gamma(q,a) \\
&= \sum_{t\in\mathcal{T}} \tau(t) \\
&= 1
\end{aligned}$$

(the second-to-last equality holds by fairness condition on γ and the last one because τ is a distribution).

In this way, the two following random variables will have the same distribution:

- $\mathrm{Capital}(\mathcal{G}, X[0, n])$.
- $\mathrm{Capital}(\hat{\mathcal{G}}, (X \otimes T)[0, n])$ where $T \in \mathcal{T}^\omega$ is chosen at random according to τ.

[2] $\mathcal{T}$ and τ are defined in the same way as in Theorem 16.

Since X is μ-normal, by Lemma 14 again, $X \otimes T$ is $\mu \otimes \tau$-normal τ-almost surely. Thus, by Theorem 7, for τ-almost all T, $\mathrm{Capital}(\hat{\mathcal{G}}, (X \otimes T)[0, n])$ is bounded by a constant C independent on n. By the equivalence of the two random variables above, this means that almost surely, there exists a constant C such that $\mathrm{Capital}(\mathcal{G}, X[0, n]) < C$ for all n. In other words, almost surely, $\mathcal{G}$ loses on the sequence X. ◀

We remark that the dichotomy theorem for Bernoulli measures yields the following dichotomy for probabilistic μ-gamblers.

▶ **Theorem 18** (Schnorr-Stimm dichotomy theorem for PFAs). *Let X be an infinite sequence in A^ω and μ a Bernoulli measure.*

(i) *If X is μ-normal and $\mathcal{G}$ is a μ-gambler, then almost surely the capital of $\mathcal{G}$ throughout the game either is ultimately constant or decreases at an exponential rate.*

(ii) *If X is not μ-normal, then there exists a μ-gambler $\mathcal{G}$ which wins against X at an "infinitely often" exponential rate almost surely.*

5 Conclusion

The main contributions of this paper are a generalization of the Agafonov theorem for PFA with arbitrary transition probabilities which settles the open question posed by Léchine et al. [6], and an extension of Schnorr-Stimm theorem to probabilistic gamblers.

While we proved the probabilistic Agafonov theorem (Theorem 16) by reduction to the deterministic setting, it is also possible to follow with a more direct approach (i.e., without appealing to the Seiller-Simonsen result), similar to the one followed by Carton to generalize Agafonov's theorem for DFA [3]. This however makes the argument somewhat more complicated.

An interesting direction for future research is to explore whether the "uselessness of randomness" also holds for pushdown automata. These are a more powerful model of computation and indeed some normal sequences can be predicted by pushdown automata (some in a rather dramatic way, as proven by Carton and Perifel [4][3]). We can for example ask: If some probabilistic pushdown selector selects a biased subsequence from a sequence X, does there necessarily exist a deterministic pushdown selector which also selects a biased subsequence? Similarly, if some probabilistic pushdown gambler wins against a sequence X, does there necessarily exist a deterministic pushdown gambler which wins against that same sequence X?

A related question concerns the speed of success in the gambling model. In the case of finite-state automata, Schnorr and Stimm proved that either a sequence X cannot be predicted or some gambler wins on it at an exponential rate. This dichotomy no longer holds for pushdown automata, but one may ask the following question: If some probabilistic pushdown gambler wins at an exponential rate against a sequence X, does there necessarily exist a pushdown gambler which wins against that sequence X at an exponential rate?

[3] The result proven by Carton and Perifel considers a slightly different paradigm, namely compression (which we did not discuss in this paper) instead of prediction.

References

1. V. N. Agafonov. Normal sequences and finite automata. *Soviet Mathematics Doklady*, 9:324–325, 1968.
2. Laurent Bienvenu, Valentino Delle Rose, and Tomasz Steifer. Probabilistic vs deterministic gamblers. In *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022*, volume 219 of *LIPIcs*, pages 11:1–11:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.STACS.2022.11.
3. Olivier Carton. A direct proof of Agafonov's theorem and an extension to shift of finite type. *CoRR*, abs/2005.00255, 2020. arXiv:2005.00255.
4. Olivier Carton and Sylvain Perifel. Deterministic pushdown automata can compress some normal sequences. *Logical Methods in Computer Science*, Volume 20, Issue 3, 2024.
5. T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition edition, 2006.
6. Ulysse Léchine, Thomas Seiller, and Jakob Grue Simonsen. Agafonov's theorem for probabilistic selectors. In *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024*, volume 306 of *LIPIcs*, pages 67:1–67:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.67.
7. John E. Maxfield. A short proof of Pillai's theorem on normal numbers. *Pacific Journal of Mathematics*, 2(1):23–24, 1952.
8. Satyadev Nandakumar, Subin Pulari, Prateek Vishnoi, and Gopal Viswanathan. An analogue of Pillai's theorem for continued fraction normality and an application to subsequences. *Bulletin of the London Mathematical Society*, 53(5):1414–1428, 2021.
9. Claus Schnorr and Hermann Stimm. Endliche Automaten und Zufallsfolgen. *Acta Informatica*, 1(4):345–359, 1972. doi:10.1007/BF00289514.
10. Thomas Seiller and Jakob Grue Simonsen. Agafonov's theorem for finite and infinite alphabets and probability distributions different from equidistribution. *Ergodic Theory and Dynamical Systems*, 45(11):3506–3539, 2025.

Cutoff for the Swendsen–Wang Dynamics on the Complete Graph

Antonio Blanca
Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA

Zhezheng Song
Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA

Abstract

We study the speed of convergence of the Swendsen–Wang (SW) dynamics for the q-state ferromagnetic Potts model on the n-vertex complete graph, known as the mean-field model. The SW dynamics was introduced as an attractive alternative to the local Glauber dynamics, often offering faster convergence rates to stationarity in a variety of settings. A series of works have characterized the asymptotic behavior of the speed of convergence of the mean-field SW dynamics for all $q \geq 2$ and all values of the inverse temperature parameter $\beta > 0$. In particular, it is known that when $\beta > q$ the mixing time of the SW dynamics is $\Theta(\log n)$. We strengthen this result by showing that for all $\beta > q$, there exists a constant $c(\beta, q) > 0$ such that the mixing time of the SW dynamics is $c(\beta, q) \log n + \Theta(1)$. This implies that the mean-field SW dynamics exhibits the cutoff phenomenon in this temperature regime, demonstrating that this Markov chain undergoes a sharp transition from "far from stationarity" to "well-mixed" within a narrow $\Theta(1)$ time window. The presence of cutoff is algorithmically significant, as simulating the chain for fewer steps than its mixing time could lead to highly biased samples.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Markov processes; Mathematics of computing → Random graphs

Keywords and phrases Markov chains, mixing times, cutoff phenomenon, Potts model, mean-field

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.17

Related Version *Full Version*: https://arxiv.org/abs/2507.20482 [10]

Funding *Antonio Blanca*: Research supported in part by NSF CAREER grant 2143762.

Acknowledgements This work transpired from a discussion between the first author, Andreas Galanis, Reza Gheissari, Eric Vigoda, and Daniel Štefankovič during the Workshop on Algorithms and Randomness at Georgia Tech in 2018 in which the high-level idea of the proof was outlined.

1 Introduction

The q-state ferromagnetic Potts model is a classical spin system model central in statistical physics, theoretical computer science, and discrete probability. Given a finite graph $G = (V, E)$, a set of spins or colors $\mathcal{S} = \{1, \ldots, q\}$, and an edge interaction parameter $\beta > 0$, it is defined as a probability distribution over the configurations in $\Omega = \{1, \ldots, q\}^V$, which correspond to the spin assignments from $\mathcal{S}$ to the vertices of G. Formally, the probability of a configuration $\sigma \in \Omega$ is given by:

$$\mu(\sigma) = \frac{1}{Z} \cdot e^{\beta M(\sigma)}, \tag{1}$$

© Antonio Blanca and Zhezheng Song;
licensed under Creative Commons License CC-BY 4.0

where $M(\sigma)$ denotes the number of edges of G with the same spin in both of its endpoints in σ (i.e., the monochromatic edges of G in σ), and Z is the normalizing constant or partition function. The edge interaction parameter β is proportional to the inverse temperature in physical applications; the $q = 2$ case corresponds to the classical ferromagnetic Ising model.

The algorithmic problem of efficiently generating approximate samples from μ has attracted much attention in theoretical computer science. For the Ising model (i.e., for $q = 2$), seminal results from the 1990s [30, 39] provide a polynomial-time approximate sampling algorithm for μ. Additional approximate-sampling algorithms have been discovered since then for the ferromagnetic Ising setting [15, 24], but all have a large running time of at least $O(n^{10})$ where $n = |V|$. For $q \geq 3$, the sampling problem is computationally hard, specifically #BIS-hard [22, 20], but efficient approximate sampling algorithms are still possible for many interesting families of graphs; see [25, 29, 13, 12, 1].

An algorithm with the potential of providing both faster algorithms for the $q = 2$ case and a technology for understanding the feasibility boundary of approximate sampling for $q \geq 3$ is the *Swendsen–Wang (SW) dynamics* [40]. This sophisticated Markov chain exploits the connection between the ferromagnetic Potts model and the random-cluster model [18] and was designed to bypass some of the key difficulties associated with sampling from μ at low temperatures (large β). In particular, the SW dynamics is conjectured to converge to μ in $O(n^{1/4})$ steps when $q = 2$ on any graph G for any $\beta > 0$. This Markov chain was also recently used to the expand the graph families for which approximately sampling can be efficiently done for $q \geq 3$ and β large [1].

The SW dynamics transitions from a configuration $\sigma_t \in \Omega$ to $\sigma_{t+1} \in \Omega$ as follows:

1. Independently, for every $e = \{u, v\} \in E$ if $\sigma_t(u) = \sigma_t(v)$ include e in M_t with probability $p = 1 - e^{-\beta}$;
2. Independently, for every connected component $\mathcal{C}$ in (V, M_t), draw a spin $s \in \{1, \ldots, q\}$ uniformly at random, and set $\sigma_{t+1}(v) = s$ for all $v \in \mathcal{C}$.

The SW dynamics is ergodic and reversible with respect to μ and thus converges to it [40, 18]. We refer to Step 1 as the percolation step and to Step 2 as the coloring step.

The SW dynamics is a highly non-local Markov chain, updating and potentially changing the spin of each vertex in each step, making its analysis quite difficult in most settings. Still, significant progress has been made in understanding the speed of convergence of this dynamics in various geometries, including the complete graph [16, 23, 34, 19, 21], finite subsets of the d-dimensional integer lattice graph $\mathbb{Z}^d$ [41, 9, 4, 38, 3], trees [27, 5], random graphs [7, 14, 8], locally-tree-like graphs [1], graphs with sub-exponential growth [1], graphs of bounded [6, 2] or unbounded degree [11], as well as general graphs when $q = 2$ [24]. A setting that has attracted particular interest and that will be the focus of this work is the complete graph, also known as the *mean-field model*.

On the complete graph, it is natural to re-parametrize the model and use β/n instead of β; i.e., $\mu(\sigma) \propto \exp(\frac{\beta}{n} \cdot M(\sigma))$. A detailed connection between the phase transition of the mean-field model, which we describe in detail in Remark 2, and the *mixing time* of the SW dynamics emerged from [34, 19, 21]. The mixing time of a Markov chain is defined as the number of steps until the Markov chain is close in total variation distance to its stationary distribution starting from the worst possible initial configuration, and it is the most standard notion of speed of convergence to stationarity. Formally, if P is the transition matrix of the chain, for $\varepsilon \in (0, 1)$,

$$\tau_{\text{mix}}(\varepsilon) = \max_{\sigma \in \Omega} \min_{t \geq 0} \{\|P^t(\sigma, \cdot) - \mu(\cdot)\|_{\text{TV}} \leq \varepsilon\},$$

and by convention $\tau_{\text{mix}} := \tau_{\text{mix}}(1/4)$.

When $q \geq 3$, the mixing time $\tau_{\text{mix}}^{\text{SW}}$ of the SW dynamics satisfies:

$$\tau_{\text{mix}}^{\text{SW}} = \begin{cases} \Theta(1) & \text{if } \beta < \beta_{\text{L}}; \\ \Theta(n^{1/3}) & \text{if } \beta = \beta_{\text{L}}; \\ e^{\Omega(n)} & \text{if } \beta \in (\beta_{\text{L}}, \beta_{\text{R}}); \\ \Theta(\log n) & \text{if } \beta \geq \beta_{\text{R}}; \end{cases} \tag{2}$$

see [19, 21]. It is known that $\beta_{\text{R}} = q$, but β_{L} does not have a closed form, and it is given instead implicitly as the root of a certain equation [19, 37]. When $q = 2$, $\beta_{\text{L}} = \beta_{\text{R}} = q$ and there is no slow-mixing window, and the mixing time of the mean-field SW dynamics for $q = 2$ is $\Theta(1)$ for $\beta < q$, $\Theta(n^{1/4})$ at the critical point $\beta = q$, and $\Theta(\log n)$ for $\beta > q$ [34].

In this work, we provide sharper results for the mixing time of the mean-field SW dynamics in the $\beta > \beta_{\text{R}} = q$ regime, further refining our understanding of this dynamics. We show that there exists a constant $c(\beta, q) > 0$ such that $\tau_{\text{mix}}^{\text{SW}} = c(\beta, q) \log n + \Theta(1)$, which implies that this Markov chain exhibits *cutoff*. A Markov chain exhibits cutoff if there is a sharp drop in the total variation distance from close to 1 to close 0 in an interval of time of smaller order than τ_{mix} which is called the *cutoff window*; i.e., we say that there is cutoff if as $n \to \infty$, for any fixed $\varepsilon \in (0, 1)$,

$$\tau_{\text{mix}}(\varepsilon) - \tau_{\text{mix}}(1 - \varepsilon) = o(\tau_{\text{mix}}).$$

We can now state our main result.

▶ **Theorem 1.** *Let $q \geq 2$ and $\beta > \beta_{\text{R}} = q$. Then, there exists a constant $c(\beta, q) > 0$ such that SW dynamics for the q-state mean-field ferromagnetic Potts model exhibits cutoff at mixing time $\tau_{\text{mix}}^{\text{SW}} = c(\beta, q) \log n$ with cutoff window $\Theta(1)$; that is, $\tau_{\text{mix}}^{\text{SW}} = c(\beta, q) \log n + \Theta(1)$.*

We provide the expression for the constant $c(\beta, q)$ in (4).

The only previously known mixing time cutoff result for the SW dynamics was on the integer lattice graph for sufficiently small enough β [38]; ours is the first such result in the low-temperature SW dynamics. For the local Glauber dynamics, which updates a single randomly chosen vertex in each step, cutoff is known in the mean-field through the high-temperature $\beta < \beta_{\text{L}}$ regime [32, 17]. For the special case of the Ising model, cutoff has also been established for the Glauber dynamics for several graph families; see [35, 36, 31, 42].

Recall from (2) that the mixing time of the SW dynamics is $\Theta(1)$ when $\beta < \beta_{\text{L}}$ and exponentially slow when $\beta \in (\beta_{\text{L}}, \beta_{\text{R}})$, so the question of whether the chain exhibits cutoff is of limited significance in those parameter regimes. It is an interesting open question, however, whether the SW dynamics exhibits cutoff at the dynamical critical points $\beta = \beta_{\text{L}}$ or $\beta = \beta_{\text{R}}$. In the former, mixing is not entirely governed by a "strong drift" towards a typical spin count distribution and relies on the variance of the dynamics as well; as such, it is unlikely that the dynamics exhibits cutoff at $\beta = \beta_{\text{L}}$. For $\beta = \beta_{\text{R}}$, the SW dynamics exhibits a first-order drift towards typical configurations, but the percolation step is critical, which significantly complicates even the asymptotic mixing time analysis.

From an algorithmic perspective, for a Markov chain like the SW dynamics that exhibits cutoff at $c(\beta, q) \log n$, knowing only the asymptotic mixing time, e.g., that $\tau_{\text{mix}}^{\text{SW}} = \Theta(\log n)$, is insufficient in practice, since simulating the dynamics for say $\frac{c(\beta,q)}{2} \log n$ steps, is guaranteed to result in a sample with total variation distance close to 1.

We conclude this introduction with some brief remarks about our proof and techniques. We construct a multi-phase coupling that converges to a pair of configurations with roughly the correct (expected) number of vertices in each spin class. (In the mean-field setting when $\beta > \beta_{\text{R}}$, under the stationary distribution μ, there is one dominant spin and all other spins

are assigned to roughly the same number of vertices.) Our coupling gradually contracts the distance to this type of configuration over two phases: first to a linear distance and then to within $O(\sqrt{n})$ distance. The second phase determines the leading order of the mixing time. Earlier analyses (i.e., [19]) yield the $O(\log n)$ mixing time bound but treat step deviations "pessimistically" and cannot provide the sharper bound we derive here. Our analysis carefully accounts for how deviations are amortized over time and account for the fact that early fluctuations do not matter as much.

An innovation of the present argument is the use of a $q \times q$ projection to a partition $\{V_1, \dots, V_q\}$ of V at *low temperatures*. This projection contains information about the number of vertices assigned each spin in each V_i and its convergence is known to imply that of the SW dynamics due to the symmetry of the mean-field model. This idea was used before in [17, 34, 19] in the high-temperature $\beta < \beta_{\mathrm{L}}$ setting where the percolation steps of the SW dynamics result in subcritical random (sub)graphs with trivial connected component structures. In the $\beta > \beta_{\mathrm{R}}$ regime, we must account for the existence of a linear-sized connected component and how it is distributed in the partition $\{V_1, \dots, V_q\}$. To address this, we have an additional step in our analysis in which we argue that the fraction of vertices assigned each spin in each V_i coalesces to the overall fraction of vertices with that spin in the full configuration before the full configuration reaches a typical spin count (i.e., before the end of second phase of the coupling). This allows us to analyze the final phase of our coupling.

▶ Remark 2. As mentioned earlier, the mixing behavior of the SW dynamics described in (2) is tightly connected to the order-disorder phase transition of the mean-field model which we describe next for completeness. This phase transition occurs at the critical value $\beta = \beta_{\mathrm{CR}}(q)$, where $\beta_{\mathrm{CR}}(q) = q$ when $q = 2$ and

$$\beta_{\mathrm{CR}}(q) = 2\Big(\frac{q-1}{q-2}\Big)\log(q-1)$$

for $q \geq 3$. When $\beta < \beta_{\mathrm{CR}}(q)$, the number of vertices assigned each spin is roughly the same with high probability up to lower-order fluctuations. In contrast, when $\beta > \beta_{\mathrm{CR}}(q)$ there is a dominant spin in the configuration with high probability. For $q \geq 3$, the model exhibits *phase coexistence* at the critical threshold $\beta = \beta_{\mathrm{CR}}(q)$; this means that at this point, the set of configurations with no dominant spin and the set of configurations with a dominant spin (with all other non-dominant spins assigned to roughly the same number of vertices), contribute each a constant fraction of the probability mass with all other configurations contributing 0 mass (in the limit as $n \to \infty$). The effects of the phase coexistence phenomenon extends to the window $(\beta_{\mathrm{L}}, \beta_{\mathrm{R}})$ around $\beta_{\mathrm{CR}}(q)$; this window is known as the metastability window and coincides with slow-mixing regime for the SW dynamics. For $q = 2$, on the other hand, there is no phase coexistence at $\beta_{\mathrm{CR}}(q)$ or metastability, so there is no slow mixing.

2 Preliminaries

We gather here a number of standard definitions and results that we will use in our proofs.

2.1 Couplings

A one step coupling of the Markov chain with state space Ω specifies for every pair of states (X_t, Y_t) a probability distribution over (X_{t+1}, Y_{t+1}) such that the processes $\{X_t\}$ and $\{Y_t\}$ are each faithful realizations of the chain, and if $X_t = Y_t$ then $X_{t+1} = Y_{t+1}$. The *coupling time* is defined by

$$T_{\mathrm{coup}} = \max_{x,y\in\Omega} \min_t \{X_t = Y_t \mid X_0 = x, Y_0 = y\}.$$

It is a standard fact that if $\Pr(T_{\text{coup}} > t) \leq \varepsilon$, then $\tau_{\text{mix}}(\varepsilon) \leq t$; see [33]. In addition, to bound $\tau_{\text{mix}} = \tau_{\text{mix}}(1/4)$, it suffices to bound $\Pr(T_{\text{coup}} > t)$ by any constant $\varepsilon < 1$, as the bound can then be boosted by repeating the coupling a constant number of times. For establishing cutoff, however, this approach is not feasible, as it would increase the mixing time bound by a multiplicative factor. Throughout our proofs, we track the probability of success of each phase of our multi-phase coupling so that $\Pr(T_{\text{coup}} > t) \leq 1/4$.

2.2 Random graph lemmas

Let $\mathcal{G}$ be a random graph distributed as a $\mathcal{G}(n, \lambda/n)$ random graph with $\lambda > 0$. Let $L_i(\mathcal{G})$ denote the size of the i-th largest component of $\mathcal{G}$ (breaking ties arbitrarily) and let $C(v)$ denote the connected component of vertex v in $\mathcal{G}$.

▶ **Lemma 3** (Lemma 5.7 [34]). *Let I be the number of isolated vertices in $\mathcal{G}$. For any constant $\lambda > 0$, there exists a constant $C > 0$ that $\Pr(I \geq Cn) = 1 - O(n^{-1})$.*

For $\lambda > 1$ let $\theta(\lambda)$ denote the unique positive solution of the equation

$$e^{-\lambda x} = 1 - x. \tag{3}$$

▶ **Lemma 4** (Lemma 5.4 [34]). *For $\lambda > 1$, there exist constants $c = c(\lambda) > 0$ and $C > 0$ such that for any $A > 0$*

$$\Pr(|L_1(\mathcal{G}) - \theta(\lambda)n| \geq A\sqrt{n}) \leq Ce^{-cA^2}.$$

Let $\mathcal{L}_1(\mathcal{G})$ denote the largest component of $\mathcal{G}$. Lemma 4 and Hoeffding's inequality, which also holds when sampling without replacement (see [26]) imply the following.

▶ **Lemma 5.** *Let $U \subseteq V$. For $\lambda > 1$, there exist constants $c = c(\lambda) > 0$ and $C > 0$ such that for any $A > 0$*

$$\Pr(||\mathcal{L}_1(\mathcal{G}) \cap U| - \theta(\lambda)|U|| \geq A\sqrt{n}) \leq Ce^{-cA^2}.$$

Finally, Theorem 1.1 in [28] and the reflection principle yields the following lemma.

▶ **Lemma 6.**
1. *If $\lambda < 1$, $\exists \gamma = \gamma(\lambda)$ such that $\mathrm{E}[\sum_{j\geq 1} L_j(\mathcal{G})^2] = \gamma n + O(1)$ and $\mathrm{Var}[\sum_{j\geq 1} L_j(\mathcal{G})^2] = O(n)$.*
2. *If $\lambda > 1$, $\exists \gamma = \gamma(\lambda)$ such that $\mathrm{E}[\sum_{j\geq 2} L_j(\mathcal{G})^2] = \gamma n + O(1)$ and $\mathrm{Var}[\sum_{j\geq 2} L_j(\mathcal{G})^2] = O(n)$.*

2.3 Drift function

Let us define the function $F : [1/q, 1] \to [0, 1]$ as

$$F(x) = \frac{1}{q} + \Big(1 - \frac{1}{q}\Big)\theta(\beta x)x,$$

with $\theta(\beta x)$ given by (3). This function plays a central role in the analysis of the mixing time of the SW dynamics, as it captures the expected fraction of vertices in the largest color class after one step of the SW dynamics from a configuration with an x fraction of the vertices in its largest color class. As such, we refer to F as the "drift function". Our proofs will use the following facts about F; the first two were previously established in [19] and we provide proofs of the next two in the full version of this paper [10].

▶ **Fact 7** (Lemma 4 [19]). *For $\beta > q$, F has a unique fixed point $a(\beta, q) \in (1/q, 1]$ and $|F'(a(\beta, q))| < 1$.*

▶ **Fact 8** (Lemma 8 [19]). *For $\beta > 0$, $F'(x) > 0$ and $F''(x) < 0$ for all $x \in (1/\beta, 1]$. That is, F is strictly increasing and concave in the interval $(1/\beta, 1]$.*

▶ **Fact 9.** *There exists $\Delta(q,\beta) > 0$ such that $|F''(x)| \le \Delta(q,\beta)$ for all $x \in (0,1]$.*

▶ **Fact 10.** $F'(a) = \frac{q-1}{q} \cdot \frac{\theta(a\beta)^2}{\theta(a\beta)+(1-\theta(a\beta))\log(1-\theta(a\beta))} \ge \frac{q-1}{q}$.

3 Proof of main result: cutoff when $\beta > q$

We assume through the remainder of the paper that $\beta > \beta_{\mathrm{R}} = q$. Let $a = a(\beta, q) > 1/q$ be the unique fixed point of the drift function F in $(1/q, 1]$; see Fact 7. We use in our analysis two different low-dimensional projections of the SW dynamics. The first is the q-dimensional *proportions* vector. For a configuration $X \in \Omega$, let $\alpha(X)$ be a q-dimensional vector, where each coordinate of $\alpha(X)$ corresponds to the fraction of vertices assigned a given spin in X. We will assume that $\alpha_1(X)$ contains the fraction of vertices assigned the dominant spin in X, breaking ties arbitrarily; that is, $\alpha_1(X) \ge \alpha_i(X)$ for all $i \in [q]$.

When $\beta > \beta_{\mathrm{R}}$, for a configuration $X \sim \mu$, $\alpha(X)$ will concentrate around the q-dimensional *majority* vector

$$m := \left(a, \frac{1-a}{q-1}, \ldots, \frac{1-a}{q-1}\right)$$

or one of its permutations.

We will design a multi-phase coupling, and in the first two phases, we bound the time it takes a copy of the SW dynamics to reach the neighborhood of m from an arbitrary initial configuration. We use the first phase to argue that the SW dynamics reaches constant distance from m and the second phase to further contract the distance to $O(1/\sqrt{n})$; the latter will dominate the order of the mixing time.

Let $X_0 \in \Omega$ be an arbitrary configuration and $Y_0 \sim \mu$. We will bound the probability that $\{X_t\}$ and $\{Y_t\}$ have not coupled after $c(\beta,q)\log n + O(1)$ steps, where

$$c(\beta,q) = \left(2\log\left(\frac{1}{F'(a)}\right)\right)^{-1} = \frac{1}{2\log\left(\frac{q}{q-1}\cdot\frac{\theta(a\beta)+(1-\theta(a\beta))\log(1-\theta(a\beta))}{\theta(a\beta)^2}\right)}; \tag{4}$$

see Fact 10. From (3), we know that $\theta(a\beta)$ is the unique root in $(0,1]$ of the equation

$$e^{\frac{-\beta y}{q-(q-1)y}} = 1 - y.$$

In the first two phases $\{X_t\}_{t\ge 0}$, $\{Y_t\}_{t\ge 0}$ evolve independently. We claim that after $T_1 = O(1)$ steps $\|\alpha(X_{T_1}) - m\|_\infty \le \varepsilon$ and $\|\alpha(Y_{T_1}) - m\|_\infty \le \varepsilon$ for any desired constant $\varepsilon \in (0,1)$.

▶ **Lemma 11.** *For any constants $\delta \in (0,1)$ and $\varepsilon > 0$ sufficiently small, for all sufficiently large n and any starting state $X_0 \in \Omega$, after $T = O(1)$ steps, with probability at least δ, the SW dynamics reaches a state X_T such that $\|\alpha(X_T) - m\|_\infty \le \varepsilon$.*

A version of this lemma was established in [19], but we require a slight refinement here so that the probability of success is any constant arbitrarily close to 1.

In the second phase, we claim that after an additional $T_2 = c(\beta,q)\log n + O(1)$ steps, in which both copies of the chain continue to evolve independently, for any constant $\delta \in (0,1)$ there exists a constant $C = C(\delta) > 0$ such that with with probability at least δ:

$$\|\alpha(X_{T_1+T_2}) - m\|_\infty \le \frac{C}{\sqrt{n}}, \text{ and } \|\alpha(Y_{T_1+T_2}) - m\|_\infty \le \frac{C}{\sqrt{n}}. \tag{5}$$

This is established in the following lemma.

▶ **Lemma 12.** *Suppose X_0 is such that $\|\alpha(X_0) - m\|_\infty \le \varepsilon$ for a sufficiently small constant $\varepsilon > 0$ and let $T = c(\beta, q)\log n + \gamma$ for constant $\gamma > 0$. Then, for any constant $\delta \in (0,1)$, there exists a constant $C = C(\gamma, \delta) > 0$ such that with probability at least δ, we have $\|\alpha(X_T) - m\|_\infty \le C/\sqrt{n}$.*

A similar convergence result was proved in [19], where it is required that $T = A\log n$ with $A > c(\beta, q)$. This allowed for a simpler analysis. To obtain our sharper bound, we use the fact that deviations from the expected proportions vector after one step are amortized over the remaining steps of the dynamics, so that early (larger) deviations end up contributing less to the overall separation from m.

In the analysis of the third phase of coupling, we use an additional "stationary-like" property for both copies of the chain. Let $\mathcal{V} = \{V_1, \dots, V_q\}$ be a partition of V where V_i contains all vertices of X_{T_1} assigned spin i. For constant $\lambda \in (0,1)$, we say that $\mathcal{V}$ is a λ-partition if $\min_i |V_i| \ge \lambda |V|$. From Lemma 11 we know that $\mathcal{V}$ is a λ-partition with constant probability arbitrarily close to 1.

Let $\alpha_{1,i}(X_t)$ denote the fraction of vertices of V_i assigned the dominant spin in X_t; i.e., there are $\alpha_{1,i}(X_t)|V_i|$ such vertices. The following lemma ensures that during the second phase, in parallel to making progress towards m, at some time $T_1 + T_2$, the fraction of vertices assigned the spin from the dominant spin class in each V_i is roughly the same fraction as that from the full configuration X_T in V. This additional property will allow us to couple the two process in $O(1)$ additional steps in the third phase.

▶ **Lemma 13.** *Let X_0 be an arbitrary initial configuration and let $\mathcal{V}$ be a λ-partition of V with constant $\lambda \in (0,1)$. For any $\delta \in (0,1)$, there exist a constant $C > 0$ such that with probability at least δ when $T \ge \frac{\log n}{2\log\left(\frac{q}{q-1}\right)} + \frac{\log(2/C)}{\log\left(\frac{q}{q-1}\right)}$, we have $\max_{k\in[q]} |\alpha_1(X_T) - \alpha_{1,k}(X_T)| \le C/\sqrt{n}$.*

We note that $(2\log\left(\frac{q}{q-1}\right))^{-1} < c(\beta, q)$ by Fact (10), and so at time $T_1 + T_2$ we reached two configurations that are close to m, i.e., satisfy (5), but that also have $\approx a|V_i|$ vertices assigned the dominant spin in each partition set V_i.

For the third phase, we consider another lower-dimensional projection of the SW dynamics; in particular, we zoom into a projection of dimension $q \times q$. For $\{X_t\}_{t\ge 0}$, let $A_t := \{A_{ij,t}\}_{i,j\in[q]}$, with $A_{ij,t}$ denoting the number of vertices in V_i assigned color j in X_t. Similarly, define A'_t for $\{Y_t\}_{t\ge 0}$.

We have shown that after the first two phases of the coupling, for any constant $\hat{\delta} \in (0,1)$ there is a suitable constant $\hat{C} > 0$ such that at time $T = T_1 + T_2$ with probability at least $\hat{\delta}$ the following holds:

A1. $\|\alpha(X_T) - m\|_\infty \le \hat{C}/\sqrt{n}$ and $\|\alpha(Y_T) - m\|_\infty \le \hat{C}/\sqrt{n}$;
A2. $|\alpha_{1,i}(X_T) - \alpha_{1,i}(Y_T)| \le \hat{C}/\sqrt{V_i}$ for all $i \in [q]$.

Under these assumptions, we design a coupling such that $A_{T+1} = A'_{T+1}$ with probability $\Omega(1)$.

▶ **Lemma 14.** *Let $\lambda \in (0,1)$ be a constant independent of n. Let $\mathcal{V} = \{V_1, ..., V_q\}$ be a λ-partition and let A_t and A'_t be two copies of the projection chain such that* A1 *and* A2 *above hold. Then, there exists a coupling of A_{t+1} and A'_{t+1} such that with probability $\Omega(1)$, it holds that $A_{t+1} = A'_{t+1}$.*

Similar couplings were also designed in [34, 19] but for the high-temperature $\beta < \beta_{\text{L}}$ setting where all the spin classes have roughly the same number of vertices and the percolation steps are subcritical. Here, we have to contend with the presence of a dominant spin class and a supercritical percolation step; this is the reason for the additional step in our analysis (Lemma 14) that guarantees that the extra assumption A2 holds after the first two phases.

The final step of our coupling is a simple boosting phase so that the overall probability of success is at least $\geq 3/4$.

▶ **Lemma 15.** *Let X_0 and Y_0 be configurations such that* A1 *and* A2 *hold. Then, there exists a coupling and a constant integer $\ell \geq 0$ such that* $\Pr\left(\cup_{i=1}^{\ell}\{A_{t+i} = A'_{t+i}\}\right) \geq \frac{3}{4}$.

In summary, we have shown that there exists a $T = c(\beta, q)\log n + O(1)$ such that $\|A_T - A'_T\|_{\text{TV}} \leq 1/4$. It was already noted in [34, 19] that the symmetry of the mean-field setting implies that $\|X_T - Y_T\| = \|A_T - A'_T\|_{\text{TV}}$ and thus

$$\tau_{\text{mix}}^{\text{SW}} \leq T = c(\beta, q)\log n + O(1),$$

which completes the proof of the upper bound in Theorem 1. The lower bound in this theorem follows straightforwardly from some of the same facts required to establish Lemma 12, and it is provided later in Appendix A.

We comment briefly in the organization of the rest of the paper. The proofs of Lemmas 11 and 12, as well as that of the mixing time lower bound, are provided in Section 4. Section 5 contains the proof of Lemma 13, and Section 6 those of Lemmas 14 and 15.

4 Convergence to the neighborhood of the majority vector

We provide next the the proofs of Lemmas 11 and 12, which ensure that SW dynamics reaches an $O(1/\sqrt{n})$ ball around m after $T = c(\beta, q)\log n + O(1)$ steps. For $b \in (0, 1)$, let

$$\kappa(b) := \Big(b, \frac{1-b}{q-1}, \ldots, \frac{1-b}{q-1}\Big).$$

Let $F^k(x) = F^{k-1}(F(x))$ with $F^1 = F$. We show first that $\alpha(X_t)$ is close to $\kappa(F^t(\alpha_1(X_0)))$ for $t \geq 0$ and then that, for an appropriate choice of t, $\kappa(F^t(\alpha_1(X_0)))$ is close to m.

▶ **Lemma 16.** *Suppose X_0 is such that $\|\alpha(X_0) - m\|_\infty \leq \varepsilon$ for a sufficiently small constant $\varepsilon > 0$. Then, there exists a constant $\hat{c}(\beta, q) > c(\beta, q)$, such that for any $t \leq \hat{c}(\beta, q)\log n$ and any constant $\delta \in (0, 1)$, there exists a constant $C(\delta) > 0$ such that with probability at least δ*

$$\|\alpha(X_t) - \kappa(F^t(\alpha_1(X_0)))\|_\infty \leq \frac{C}{\sqrt{n}}.$$

▶ **Lemma 17.** *Let $x_0 \in (a - \varepsilon, a + \varepsilon)$ where $\varepsilon > 0$ is small enough and let $\gamma \in \mathbb{R}$. If $T = c(\beta, q)\log n + \gamma$, there exist $C_1 = C_1(\beta, q, \varepsilon) > 0$ and $C_2 = C_2(\beta, q, \varepsilon) > 0$ such that*

$$\frac{|x_0 - a|F'(a)^\gamma C_1}{\sqrt{n}} \leq |F^T(x_0) - a| \leq \frac{|x_0 - a|F'(a)^\gamma C_2}{\sqrt{n}}.$$

A key step in the proof of Lemma 16, which is also useful for proving Lemma 11 is the following concentration guarantee on the fluctuations.

▶ **Lemma 18.** *Suppose X_t is such that $\|\alpha(X_t) - m\|_\infty \leq \varepsilon$ for a sufficiently small $\varepsilon > 0$. Then, there exists $\gamma_0(\beta)$ such that for any $r \geq \gamma_0(\beta)$*

$$\Pr\left(\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \geq \frac{r}{\sqrt{n}}\right) \leq e^{-\Omega(r)} + O(r^{-2}).$$

Before proving these lemmas, we show how use them to establish Lemmas 11 and 12.

Proof Lemma 11. From Lemma 30 in [19], we have that for any constant $\varepsilon > 0$ and any starting state $X_0 \in \Omega$, there exists $T = O(1)$ such that after T steps, we have that $\|\alpha(X_T) - m\|_\infty \leq \varepsilon$ with probability at least $\gamma = \Omega(1)$. Running the dynamics for kT steps, ensures the probability that we do not reach a configuration X such that $\|\alpha(X) - m\|_\infty \leq \varepsilon$ after kT steps is at most $(1-\gamma)^k$.

To complete the proof, we we show that once we reach a configuration $\|\alpha(X_t) - m\|_\infty \leq \varepsilon$, then this property is preserved for many steps with high probability. In particular, we show that X_{t+1} is such that $\|\alpha(X_{t+1}) - m\|_\infty \leq \varepsilon$ with probability $1 - O(n^{-1})$ provided ε is small enough. The result then follows from a union bound over the required $O(1)$ steps.

If $\|\alpha(X_t) - m\|_\infty \leq \varepsilon$, then $\alpha_1(X_t) \in [a-\varepsilon, a+\varepsilon]$. By Fact 7, for ε small enough there exists a constant $\rho \in (0,1)$ such that $|F(a+\varepsilon) - a| \leq \rho\varepsilon$ and $|F(a-\varepsilon) - a| \leq \rho\varepsilon$. Since F is increasing and concave in $(1/q, 1)$ (see Fact 8), this implies that $(a+\varepsilon) - F(a+\varepsilon) \geq (1-\rho)\varepsilon$ and $F(a-\varepsilon) - (a-\varepsilon) \geq (1-\rho)\varepsilon$. Thus, in order for $\|\alpha(X_{t+1}) - m\|_\infty > \varepsilon$, we would need $|\alpha_1(X_{t+1}) - F(\alpha_1(X_t))| \geq (1-\rho)\varepsilon$, which implies that $\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \geq (1-\rho)\varepsilon$. Then, from Lemma 18 we have that

$$\Pr\Big(\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \geq (1-\rho)\varepsilon\Big) \leq O\big(n^{-1}\big). \qquad \blacktriangleleft$$

Proof of Lemma 12. Follows from Lemmas 16 and 17 and the triangle inequality. ◀

4.1 Proofs of Lemmas 16, 17, and 18

We provide first the proof of and Lemma 18.

Proof of Lemma 18. Since $\beta > q$ and $a > 1/q$ by Fact 7, $a\beta > 1$ and $\frac{1-a}{q-1}\beta < 1$. By assumption $\|\alpha(X_t) - m\|_\infty \leq \varepsilon$, so for ε small enough, exactly one color class in the percolation step is supercritical, and the remaining $q-1$ color classes are subcritical.

Let $C_1, C_2, \ldots$ denote the connected components after the percolation step from X_t, sorted in decreasing order by size (breaking ties arbitrarily) and let $R = \sum_{i \geq 2} |C_i|^2$. By Lemma 6 there exist constants $\gamma_0, \gamma_1 > 0$, that depend only on β and q, such that $\mathrm{E}[R] = \gamma_0 n + O(1)$ and $\mathrm{Var}[R] \leq \gamma_1 n$. Then, by Chebyshev's inequality, for any $\gamma_2 \geq 1$ and n large we have

$$\Pr\left(|R - \gamma_0 n| \geq \gamma_2\sqrt{\gamma_1 n}\right) \leq \Pr\left(|R - \mathrm{E}[R]| \geq \frac{\gamma_2}{2}\sqrt{\gamma_1 n}\right) \leq \frac{4}{\gamma_2^2}. \tag{6}$$

For $j \geq 2$ and $i \in [q]$, consider the random variables $Q_j^{(i)}$ where

$$Q_j^{(i)} = \begin{cases} |C_j| & \text{if } C_j \text{ colored } i, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Let $Z_i = \sum_{j \geq 2} Q_j^{(i)}$. By Hoeffding's inequality, for any $\gamma_3 \geq 0$

$$\Pr\left(|Z_i - E[Z_i]| \geq \gamma_3\sqrt{n} \mid R \leq \gamma_0 n + \gamma_2\sqrt{\gamma_1 n}\right) \leq 2\exp\left(\frac{-2\gamma_3^2}{\gamma_0 + \gamma_2\sqrt{\gamma_1/n}}\right),$$

and combined with (6) we deduce that

$$\Pr\left[|Z_i - E[Z_i]| \geq \gamma_3\sqrt{n}\right] \leq 2\exp\left(\frac{-\gamma_3^2}{\gamma_0 + \gamma_2\sqrt{\gamma_1/n}}\right) + \frac{4}{\gamma_2^2}.$$

Turning our attention to $|C_1|$, by Lemma 4, there are exists constants $c_1, c_2 > 0$ such that for any $\gamma_4 > 0$

$$\Pr(||C_1| - \theta(\beta\alpha_1(X_t))\alpha_1(X_t)n| \geq \gamma_4\sqrt{n}) \leq c_1 \exp(-c_2\gamma_4^2). \tag{7}$$

If $||C_1| - \theta(\beta\alpha_1(X_t))\alpha_1(X_t)n| < \gamma_4\sqrt{n}$, and $|Z_j - E[Z_j]| < \gamma_3\sqrt{n}$ for all $i \in [q]$, we have

$$\begin{aligned}\alpha_1(X_{t+1})n &\leq \theta(\beta\alpha_1(X_t))\alpha_1(X_t)n + \frac{n - \theta(\beta\alpha_1(X_t))\alpha_1(X_t)n}{q} + (\gamma_3+\gamma_4)\sqrt{n} \\ &= F(\alpha_1(X_t))n + (\gamma_3+\gamma_4)\sqrt{n},\end{aligned}$$

and similarly

$$\alpha_1(X_{t+1})n \geq F(\alpha_1(X_t))n - (\gamma_3+\gamma_4)\sqrt{n}.$$

In addition, for $j \neq 1$

$$\alpha_j(X_{t+1})n \leq \frac{n - \theta(\beta\alpha_1(X_t))\alpha_1(X_t)n}{q} + (\gamma_3+\gamma_4)\sqrt{n} \leq \frac{(1 - F(\alpha_1(X_t)))n}{q-1} + (\gamma_3+\gamma_4)\sqrt{n},$$

and

$$\alpha_j(X_{t+1})n \geq \frac{(1 - F(\alpha_1(X_t)))n}{q-1} - (\gamma_3+\gamma_4)\sqrt{n}.$$

Combining these facts via a union bound, and setting $\gamma_2 = \gamma_3 = \gamma_4 = r/2$, it follows that for $r \geq \gamma_0$ and n large enough that

$$\begin{aligned}\Pr\Big(\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \geq \frac{r}{\sqrt{n}}\Big) &\leq c_1 \cdot e^{-c_2\gamma_4^2} + 2q\exp\Big(\frac{-\gamma_3^2}{\gamma_0 + \gamma_2\sqrt{\gamma_1/n}}\Big) + \frac{4q}{\gamma_2^2} \\ &\leq e^{-\Omega(r)} + O(r^{-2}).\end{aligned}$$

◀

We are now ready to provide the proof of Lemma 16.

Proof of Lemma 16. Let

$$\xi_t = n \cdot \|\alpha(X_t) - \kappa(F(\alpha_1(X_{t-1})))\|_\infty. \tag{8}$$

By Fact 7, $|F'(a)| < 1$, so for small enough ε, there exists $\rho \in (0,1)$ such that $1 - \rho = \max_{|x-a|<2\varepsilon} |F'(x)|$. Let $K > 0$ be a large constant we choose later and define the sequence

$$r_i = K\big(1 - \rho/2\big)^{t-i}\big(1+\rho\big)^{t-i}. \tag{9}$$

Our first observation is that when $\xi_i < r_i\sqrt{n}$ for all $i \leq t$, we have

$$|F^{t-j-1}(\alpha_1(X_j)) - a| < 2\varepsilon \tag{10}$$

for all $j \leq t-1$. To see this, note that since F is increasing, concave and have a unique fixed point in $[1/q, 1]$ (see Facts 7 and 8) we have that for any $y \in [1/q, 1]$

$$|F(y) - a| \leq |y - a|. \tag{11}$$

Hence, $|F^{t-j-1}(\alpha_1(X_j)) - a| \leq |\alpha_1(X_j) - a|$, and from (8) and the triangle inequality it follows that

$$|\alpha_1(X_j) - a| \leq |F(\alpha_1(X_{j-1})) - a| + \frac{\xi_j}{n}.$$

Using (11) again and combining these inequalities we conclude that:

$$|F^{t-j-1}(\alpha_1(X_j)) - a| \leq |\alpha_1(X_{j-1}) - a| + \frac{\xi_j}{n}.$$

Iterating this process:

$$|F^{t-j-1}(\alpha_1(X_j)) - a| \leq |\alpha_1(X_0) - a| + \frac{1}{n}\sum_{i=1}^{j}\xi_i \leq \varepsilon + \frac{1}{\sqrt{n}}\sum_{i=1}^{j} r_i.$$

Observe that for a suitable constant $K' > 0$,

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{j} r_i = \frac{K}{\sqrt{n}}\sum_{i=1}^{t}\big(1-\rho/2\big)^{t-i}\big(1+\rho\big)^{t-i} \leq \frac{K'(\big(1-\rho/2\big)\big(1+\rho\big))^t}{\sqrt{n}}.$$

So, when t is such that $(\big(1-\rho/2\big)\big(1+\rho\big))^t \leq n^{\frac{1}{2}-\gamma}$ for a constant $\gamma \in (1/4, 1/2)$, we have

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{j} r_i = o(1),$$

and thus $|F^{t-j-1}(\alpha_1(X_j)) - a| \leq \varepsilon + o(1) \leq 2\varepsilon$ for n large which establishes (10). Observe that

$$(\big(1-\rho/2\big)\big(1+\rho\big))^t \leq n^{\frac{1}{2}-\gamma}$$

when $t \leq \frac{\frac{1}{2}-\gamma}{\log((1-\rho/2)(1+\rho))}\log n$, and it can be checked that

$$\frac{\frac{1}{2}-\gamma}{\log((1-\rho/2)(1+\rho))} > c(\beta, q) = \frac{1}{2\log\frac{1}{F'(a)}}.$$

Now, going back to (8),

$$F(\alpha_1(X_{t-1})) - \frac{\xi_t}{n} \leq \alpha_1(X_t) \leq F(\alpha_1(X_{t-1})) + \frac{\xi_t}{n}.$$

Similarly,

$$F(\alpha_1(X_{t-2})) - \frac{\xi_{t-1}}{n} \leq \alpha_1(X_{t-1}) \leq F(\alpha_1(X_{t-2})) + \frac{\xi_{t-1}}{n}, \tag{12}$$

and since F is increasing in $[1/q, 1]$ (see Fact 8), we have

$$F\Big(F(\alpha_1(X_{t-2})) - \frac{\xi_{t-1}}{n}\Big) - \frac{\xi_t}{n} \leq \alpha_1(X_t) \leq F\Big(F(\alpha_1(X_{t-2})) + \frac{\xi_{t-1}}{n}\Big) + \frac{\xi_t}{n}.$$

Since from (10) we know that $F(\alpha_1(X_{t-2})) \in (a-2\varepsilon, a+2\varepsilon)$. Hence, the Taylor expansion of the function F about $F(\alpha_1(X_{t-2}))$ yields that

$$F\Big(F(\alpha_1(X_{t-2})) + \frac{\xi_{t-1}}{n}\Big) \leq F(F(\alpha_1(X_{t-2}))) + \frac{(1-\rho)\xi_{t-1}}{n}, \tag{13}$$

and

$$F\Big(F(\alpha_1(X_{t-2})) - \frac{\xi_{t-1}}{n}\Big) \geq F(F(\alpha_1(X_{t-2}))) - \frac{(1-\rho)\xi_{t-1}}{n}. \tag{14}$$

Therefore,

$$F(F(\alpha_1(X_{t-2}))) - \frac{(1-\rho)\xi_{t-1}}{n} - \frac{\xi_t}{n} \le \alpha_1(X_t) \le F(F(\alpha_1(X_{t-2}))) + \frac{(1-\rho)\xi_{t-1}}{n} + \frac{\xi_t}{n}.$$

Iterating this argument, making use of (10) in each step, we deduce that

$$F^t(\alpha_1(X_0)) - \frac{1}{n}\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i \le \alpha_1(X_t) \le F^t(\alpha_1(X_0)) + \frac{1}{n}\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i, \tag{15}$$

so that

$$|\alpha_1(X_t) - F^t(\alpha_1(X_0))| \le \frac{1}{n}\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i. \tag{16}$$

In a similar fashion, from (8) for $j \neq 1$

$$\frac{1-F(\alpha_1(X_{t-1}))}{q-1} - \frac{\xi_t}{n} \le \alpha_j(X_t) \le \frac{1-F(\alpha_1(X_{t-1}))}{q-1} + \frac{\xi_t}{n},$$

and from (12) since F is increasing

$$\frac{1-F\big(F(\alpha_1(X_{t-2})) + \frac{\xi_{t-1}}{n}\big)}{q-1} - \frac{\xi_t}{n} \le \alpha_j(X_t) \le \frac{1-F\big(F(\alpha_1(X_{t-2})) + \frac{\xi_{t-1}}{n}\big)}{q-1} + \frac{\xi_t}{n}.$$

From (13) and (14), we then deduce that

$$\frac{1-F^2\big(\alpha_1(X_{t-2})\big)}{q-1} - \frac{(1-\rho)\xi_{t-1}}{(q-1)n} - \frac{\xi_t}{n} \le \alpha_j(X_t) \le \frac{1-F^2\big(\alpha_1(X_{t-2})\big)}{q-1} + \frac{(1-\rho)\xi_{t-1}}{(q-1)n} + \frac{\xi_t}{n},$$

and iterating as in (15):

$$\left|\alpha_j(X_t) - \frac{1-F^t\big(\alpha_1(X_{t-2})\big)}{q-1}\right| \le \frac{1}{(q-1)n}\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i. \tag{17}$$

From (16) and (17), it suffices to show that for a suitable constant $C > 0$, with probability at least $\delta \in (0,1)$

$$\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i \le C\sqrt{n}.$$

When $\xi_i < r_i\sqrt{n}$ for all $i \le t$, then

$$\sum_{i=1}^{t}(1-\rho)^{t-i}\xi_i \le K\sqrt{n}\sum_{i=1}^{t}(1-\rho)^{t-i}\big(1-\rho/2\big)^{t-i}\big(1+\rho\big)^{t-i} \le C\sqrt{n}$$

for large enough C. By Lemma 18, we can choose K large enough such that

$$\sum_{i=1}^{t}\Pr(\xi_i \ge r_i\sqrt{n}) \le \sum_{i=1}^{t}O(r_i^{-2}) \le \delta.$$

It then follows from a union bound that

$$\Pr\Big(\sum_{i=1}^{t}\xi_i(1-\rho)^{t-i} \ge C\sqrt{n}\Big) \le \sum_{i=1}^{t}\Pr(\xi_i > r_i\sqrt{n}) \le \delta. \qquad \blacktriangleleft$$

We complete this proof by proving the proof of Lemma 17.

Proof of Lemma 17. For ease of notation, set $|F'(a)| = \eta < 1$. By Fact 9, there exists $\Delta \geq |F''(x)|$ for all $x \in (a-\varepsilon, a+\varepsilon)$. Let $d_t = |a - F^t(x_0)|$. The Taylor expansion of F about a implies that

$$\eta d_{t-1} - \frac{\Delta}{2} d_{t-1}^2 \leq d_t \leq \eta d_{t-1} + \frac{\Delta}{2} d_{t-1}^2. \tag{18}$$

Iterating,

$$\eta^T d_0 - \frac{\Delta}{2\eta} \sum_{i=1}^{T} \eta^i d_{T-i}^2 \leq d_T \leq \eta^T d_0 + \frac{\Delta}{2\eta} \sum_{i=1}^{T} \eta^i d_{T-i}^2.$$

Since $d_t \leq \varepsilon$ for $t \geq 0$, for ε small enough, we can deduce from (18) that

$$(\eta - \frac{\Delta\varepsilon}{2}) d_{t-1} \leq d_t \leq (\eta + \frac{\Delta\varepsilon}{2}) d_{t-1},$$

and thus

$$d_T \leq \frac{d_0 \eta^\gamma}{\sqrt{n}} + \frac{\Delta d_0^2 (\eta + \frac{\Delta\varepsilon}{2})^{2T}}{2\eta} \sum_{i=1}^{T} \Big(\frac{\eta}{(\eta + \frac{\Delta\varepsilon}{2})^2} \Big)^i,$$

and

$$d_T \geq \frac{d_0 \eta^\gamma}{\sqrt{n}} - \frac{\Delta d_0^2 (\eta - \frac{\Delta\varepsilon}{2})^{2T}}{2\eta} \sum_{i=1}^{T} \Big(\frac{\eta}{(\eta - \frac{\Delta\varepsilon}{2})^2} \Big)^i.$$

For ε small enough, $\eta > (\eta + \frac{\Delta\varepsilon}{2})^2 > (\eta - \frac{\Delta\varepsilon}{2})^2$, and thus there exist constants $\gamma_1 = \gamma_1(\eta, \Delta, \varepsilon) > 0$ and $\gamma_2 = \gamma_2(\eta, \Delta, \varepsilon) > 0$ such that, as claimed:

$$\frac{d_0 \eta^\gamma}{\sqrt{n}} - \frac{\Delta d_0^2 \gamma_2}{2\eta} \frac{\eta^\gamma}{\sqrt{n}} = \frac{d_0 \eta^\gamma (1 - \frac{\Delta\varepsilon\gamma_2}{2\eta})}{\sqrt{n}} \leq d_T \leq \frac{d_0 \eta^\gamma}{\sqrt{n}} + \frac{\Delta d_0^2 \gamma_1}{2\eta} \frac{\eta^\gamma}{\sqrt{n}} = \frac{d_0 \eta^\gamma (1 + \frac{\Delta\varepsilon\gamma_1}{2\eta})}{\sqrt{n}}. \quad \blacktriangleleft$$

5 Spin fraction coalescence in partition

We provide in this section the proof of Lemma 13. For this, for $k \in [q]$, let $d_t^{(k)} = |\alpha_1(X_t) - \alpha_{1,k}(X_t)|$ and

$$\hat{\xi}_{t+1}^{(k)} = \Big| d_{t+1}^{(k)} - \Big(1 - \frac{1}{q}\Big) \theta(\beta\alpha_1(X_t)) d_t^{(k)} \Big|. \tag{19}$$

We will need the following concentration bound for $\hat{\xi}_t^{(k)}$.

▶ **Lemma 19.** *Let $\{V_1, \ldots, V_q\}$ be a λ-partition of V and suppose X_t is such that $\|\alpha(X_t) - m\|_\infty \leq \varepsilon$ for a sufficiently small $\varepsilon > 0$. Then, there exists $\gamma_1(\beta, q) > 0$ such that for any $r \geq \gamma_1(\beta, q)$ and any $k \in [q]$*

$$\Pr\Big(\hat{\xi}_{t+1}^{(k)} \geq \frac{r}{\sqrt{n}} \Big) \leq e^{-\Omega(r)} + O(r^{-2}).$$

Proof. Since $\beta > q$ and $a > 1/q$ by Fact 7, $a\beta > 1$ and $\frac{\beta(1-a)}{q-1} < 1$. By assumption $\|\alpha(X_t) - m\|_\infty \le \varepsilon$, so when ε is small enough, exactly one color class in the percolation step of the SW dynamics is supercritical, and the remaining $q-1$ are subcritical.

Without loss of generality, let us prove the statement for V_1 (i.e., for $k=1$), noting that the same argument applies to any other V_i. Let C_1, C_2,... denote the connected components after the percolation step from X_t, sorted in decreasing order by size, let $R = \sum_{i\ge 2} |C_i|^2$ and $R_1 = \sum_{i\ge 2} |C_i \cap V_1|^2$. By Lemma 6 there exist constants $\gamma_0, \gamma_1 > 0$ such that $\mathrm{E}[R] = \gamma_0 n + O(1)$ and $\mathrm{Var}[R] \le \gamma_1 n$. Then, by Chebyshev's inequality, for any $\gamma_2 \ge 1$ and n large we have

$$\Pr\left[R_1 \ge \gamma_0 n + \gamma_2\sqrt{\gamma_1 n}\right] \le \Pr\left[R \ge \gamma_0 n + \gamma_2\sqrt{\gamma_1 n}\right] \le \Pr\left(R \ge \mathrm{E}[R] + \frac{\gamma_2}{2}\sqrt{\gamma_1 n}\right) \le \frac{4}{\gamma_2^2}. \quad (20)$$

For $j \ge 2$ and $i \in [q]$, consider the random variables $Q_j^{(i)}$ where

$$Q_j^{(i)} = \begin{cases} |C_j \cap V_1| & \text{if } C_j \text{ is colored } i; \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Let $Z_i = \sum_{j\ge 2} Q_j^{(i)}$. By Hoeffding's inequality, for any $\gamma_3 \ge 0$:

$$\Pr\left(|Z_i - E[Z_i]| \ge \gamma_3\sqrt{n} \mid R_1 \le \gamma_0 n + \gamma_2\sqrt{\gamma_1 n}\right) \le 2\exp\left(\frac{-2\gamma_3^2}{\gamma_0 + \gamma_2\sqrt{\gamma_1/n}}\right),$$

and combined with (20) we obtain

$$\Pr\left[|Z_i - E[Z_i]| \ge \gamma_3\sqrt{n}\right] \le 2\exp\left(\frac{-\gamma_3^2}{\gamma_0 + \gamma_2\sqrt{\gamma_1/n}}\right) + \frac{4}{\gamma_2^2}.$$

We consider next $|C_1 \cap V_1|$. By Lemma 5, we have that for suitable constants $c_1, c_2 > 0$ for any $\gamma_4 > 0$

$$\Pr\left(\left||C_1 \cap V_1| - \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1|\right| \ge \gamma_4\sqrt{n}\right) \le c_1\exp\left(-c_2\gamma_4^2\right).$$

If $\left||C_1 \cap V_1| - \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1|\right| < \gamma_4\sqrt{n}$, and $|Z_j - E[Z_j]| < \gamma_3\sqrt{n}$ for all $j \in [q]$:

$$\begin{aligned}\alpha_{1,1}(X_{t+1})|V_1| &\le \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1| + \frac{|V_1| - \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1|}{q} + (\gamma_3+\gamma_4)\sqrt{n} \\ &= G(\alpha_1(X_t), \alpha_{1,1}(X_t))|V_1| + (\gamma_3+\gamma_4)\sqrt{n},\end{aligned}$$

where the function $G : [1/q, 1] \times [1/q, 1] \to [0, 1]$ is defined as

$$G(x, y) = \frac{1}{q} + \theta(\beta x)y\left(1 - \frac{1}{q}\right).$$

Similarly, we have that

$$\alpha_{1,1}(X_{t+1})|V_1| \ge G(\alpha_1(X_t), \alpha_{1,1}(X_{t+1}))|V_1| - (\gamma_3+\gamma_4)\sqrt{n},$$

and for $j \ne 1$

$$\alpha_{j,1}(X_{t+1})|V_1| \le \frac{n - \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1|}{q} + (\gamma_3+\gamma_4)\sqrt{n},$$

and

$$\alpha_{j,1}(X_{t+1})|V_1| \geq \frac{n - \theta(\beta\alpha_1(X_t))\alpha_{1,1}(X_t)|V_1|}{q} - (\gamma_3+\gamma_4)\sqrt{n}.$$

Combining these facts and setting $\gamma_2 = \gamma_3 = \gamma_4 = r/4$, for $r \geq \gamma_0$ and n large enough that

$$\begin{aligned}
&\Pr\Big(|\alpha_{1,1}(X_{t+1}) - G(\alpha_1(X_t), \alpha_{1,1}(X_t))| \geq \frac{r}{2\sqrt{n}}\Big)\\
&\leq \Pr\Big(|\alpha_{1,1}(X_{t+1}) - G(\alpha_1(X_t), \alpha_{1,1}(X_t))| \geq \frac{(\gamma_3+\gamma_4)}{\sqrt{n}}\Big)\\
&\leq \Pr\Big(|\alpha_{1,1}(X_{t+1}) - G(\alpha_1(X_t), \alpha_{1,1}(X_t))| \geq \frac{(\gamma_3+\gamma_4)\sqrt{n}}{|V_1|}\Big)\\
&\leq c_1\exp(-c_2\gamma_4^2) + 2q\exp\Big(\frac{-\gamma_3^2}{\gamma_0+\gamma_2\sqrt{\gamma_1/n}}\Big) + \frac{4q}{\gamma_2^2} \leq e^{-\Omega(r)} + O(r^{-2}). \qquad (21)
\end{aligned}$$

By Lemma 18, for large enough r

$$\Pr\Big(|\alpha_1(X_{t+1}) - F(\alpha_1(X_t))| \geq \frac{r}{2\sqrt{n}}\Big) \leq e^{-\Omega(r)} + O(r^{-2}). \qquad (22)$$

Since $d_{t+1}^{(1)} = |\alpha_1(X_{t+1}) - \alpha_{1,1}(X_{t+1})|$, $\hat{\xi}_{t+1}^{(1)} = \big|d_{t+1}^{(1)} - \big(1-\frac{1}{q}\big)\theta(\beta\alpha_1(X_t))d_t\big|$ and

$$F(x) - G(x,y) = \Big(1-\frac{1}{q}\Big)(x-y)\theta(\beta x),$$

it follows from (21) and (22) that $\Pr\big(\hat{\xi}_{t+1}^{(1)} \geq \frac{r}{\sqrt{n}}\big) \leq \exp(-\Omega(r)) + O(r^{-2})$. ◀

We can now provide the proof of Lemma 13.

Proof of Lemma 13. Since $d_t^{(k)} = |\alpha_1(X_t) - \alpha_{1,k}(X_t)|$, from (19) we have

$$d_t^{(k)} \leq \Big(1-\frac{1}{q}\Big)\theta(\beta\alpha_1(X_{t-1}))d_{t-1}^{(k)} + \hat{\xi}_t^{(k)} \leq \Big(1-\frac{1}{q}\Big)d_{t-1}^{(k)} + \hat{\xi}_t^{(k)}.$$

Iterating, we deduce that

$$d_t^{(k)} \leq \Big(1-\frac{1}{q}\Big)^t + \sum_{i=1}^{t}\Big(1-\frac{1}{q}\Big)^{t-i}\hat{\xi}_i^{(k)}.$$

Let $\hat{K} > 0$, $\hat{\rho} = 1/q$, and define the sequence $\hat{r}_i = \hat{K}\big(1-\hat{\rho}/2\big)^{t-i}\big(1+\hat{\rho}\big)^{t-i}$. By Lemma 19, for any $\delta > 0$ there exists $\hat{K}$ large enough such that

$$\sum_{i=1}^{t}\Pr\Big(\hat{\xi}_i^{(k)} \geq \frac{\hat{r}_i}{\sqrt{n}}\Big) \leq \sum_{i=1}^{t} O(\hat{r}_i^{-2}) \leq \frac{\delta}{q}.$$

If $\hat{\xi}_i^{(k)} < \hat{r}_i/\sqrt{n}$ for all $i \leq t$, then

$$\sum_{i=1}^{t}(1-\hat{\rho})^{t-i}\hat{\xi}_i^{(k)} \leq \frac{\hat{K}}{\sqrt{n}}\sum_{i=1}^{t}(1-\hat{\rho})^{t-i}\big(1-\hat{\rho}/2\big)^{t-i}\big(1+\hat{\rho}\big)^{t-i} \leq \frac{C}{2\sqrt{n}},$$

for a suitable constant $C > 0$. A union bound then implies that

$$\Pr\Big(\sum_{i=1}^{t}\hat{\xi}_i^{(k)}(1-\hat{\rho})^{t-i} \geq \frac{C}{2\sqrt{n}}\Big) \leq \sum_{i=1}^{t}\Pr\Big(\hat{\xi}_i^{(k)} > \frac{\hat{r}_i}{\sqrt{n}}\Big) \leq \frac{\delta}{q}.$$

Then, when $t \geq \frac{\log n}{2\log\left(\frac{q}{q-1}\right)} + \frac{\log(2/C)}{\log\left(\frac{q}{q-1}\right)}$, we have that $d_t^{(k)} \leq C/\sqrt{n}$ with probability at least $1-\delta/q$, and the result follows by a union bound over $k \in [q]$. ◀

6 Projection chain

In this section, we prove Lemma 14 using the following multinomial coupling from [19].

▶ **Fact 20.** *Fix integers $q \geq 2$, $m \geq 0$ and a constant $L > 0$. Let X and Y be two multinomial random variables with m trials, where each trial leads to success of one of q values with the probability of success for each value being $1/q$; i.e., $X, Y \sim \mathrm{Mult}(m; 1/q, \ldots, 1/q)$. Let $w = (w_1, \ldots, w_q) \in \mathbb{Z}^q$ such that $\sum_{i=1}^q w_i = 0$ and $\|w\|_\infty \leq L\sqrt{m}$. Then there exists a coupling of X and Y and a constant $c = c(q, L) > 0$ independent of m such that $\Pr\left(X = Y + w\right) \geq c$.*

Proof of Lemma 14. We couple A_{t+1} and A'_{t+1} as follows. The percolation step is done independently in both copies. In the coloring step, first, the same random color s is assigned to the largest connected component in both chains, and all other connected components with two or more vertices from both chains are assigned spins uniformly at random independently. In addition, in each V_i, if one copy has more isolated vertices than the other, the excess of isolated vertices in that copy are assigned spins independently. The coloring of the remaining isolated vertices will be coupled to fix the potential discrepancies created in each V_i by this initial procedure as we described next.

For $i, j \in [q]$, let $\hat{a}_{ij}$, $\hat{a}'_{ij}$ be the number of vertices assigned color j in V_i so far. We claim that there exists a constant $C > 0$ such that with probability $\Omega(1)$ it holds for $i, j \in [q]$ that

$$|\hat{a}_{ij} - \hat{a}'_{ij}| \leq C\sqrt{|V_i|}. \tag{23}$$

To prove this, let $\{C_k\}_{k\geq 1}$ be the connected components in the first chain after the percolation step; assume C_1 is the largest component. Let $C(v)$ be the component of vertex v. Note that

$$\sum_{k\geq 2} |C_k \cap V_i|^2 = \sum_{k\geq 2} \sum_{v\in C_k\cap V_i} |C_k \cap V_i| = \sum_{k\geq 2} \sum_{v\in C_k\cap V_i} |C(v) \cap V_i| \leq \sum_{v\in V_i\setminus C_1} |C(v)|.$$

For $v \notin C_1$, $E[|C(v)|] \leq A_1$ for a suitable constant $A_1 > 0$. Hence, taking expectations, it follows from Markov's inequality that

$$\Pr\left(\sum\nolimits_{k\geq 2} |C_k \cap V_i|^2 \geq 100qA_1|V_i|\right) \leq \frac{1}{100q}.$$

Let $\hat{n}_i$ be the number of vertices in V_i excluding those in C_1 and the unassigned isolated vertices. If C_1 is assigned spin s, then conditioning on $\sum_{k\geq 2} |C_k \cap V_i|^2 < 100qA_1|V_i|$, by Hoeffding's inequality, we deduce that for a large enough constant $A_2 > 0$, with probability at least $\frac{99}{100q^2}$, for any $j \in [q]$ with $j \neq s$, it holds that $\left|\hat{a}_{ij} - \frac{\hat{n}_i}{q}\right| \leq A_2\sqrt{|V_i|}$.

Similarly, we can get an analogous bound for $\hat{a}'_{ij}$, and by a union bound (23) holds for all $i, j \in [q]$ such that $j \neq s$ with probability at least $96/100$.

By Lemma 5 and assumption A2, for a suitable constant $A_3 > 0$, we have for any $i \in [q]$

$$\Pr\left(||C_1 \cap V_i| - |C'_1 \cap V_i|| \geq A_3\sqrt{n}\right) \leq \frac{1}{100q}.$$

Conditioning on this event and on $\sum_{k\geq 2} |C_k \cap V_i|^2 < 100qA_1|V_i|$, both for all $i \in [q]$, we obtain via Hoeffding's inequality that for a large enough constant $A'_2 > 0$ that for any $i \in [q]$

$$\left|\hat{a}_{is} - |C_1 \cap V_i| - \frac{\hat{n}_s}{q}\right| \leq (A'_2 + A_3)\sqrt{n}, \text{ and } \left|\hat{a}'_{is} - |C'_1 \cap V_i| - \frac{\hat{n}_s}{q}\right| \leq (A'_2 + A_3)\sqrt{n}$$

with probability at least $99/(100q)$, which implies that (23) holds for all $i \in [q]$ and $j = s$ with that probability $99/100$ by a union bound.

Since $\mathcal{V}$ is a λ-partition, Lemma 3 and a union bound over the partition sets imply that there are at least $\Omega(n)$ unassigned isolated vertices in each V_i in both chains with probability $1 - O(n^{-1})$. Observe that the distribution of the coloring of the isolated vertices in each V_i and each copy is a multinomial distribution. Then, using Fact 20, we can couple the coloring of the isolated vertices in each V_i to equalize the spin counts and obtain two configurations such that $A_{t+1} = A'_{t+1}$ with probability $\Omega(1)$. ◀

References

1 A. Blanca and R. Gheissari. Sampling from the Potts model at low temperatures via Swendsen–Wang dynamics. In *Proceedings of the 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2006–2020, 2023. doi:10.1109/FOCS57990.2023.00122.

2 Antonio Blanca, Pietro Caputo, Zongchen Chen, Daniel Parisi, Daniel Štefankovič, and Eric Vigoda. On mixing of markov chains: Coupling, spectral independence, and entropy factorization. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3670–3692, 2022. doi:10.1137/1.9781611977073.145.

3 Antonio Blanca, Pietro Caputo, Daniel Parisi, Alistair Sinclair, and Eric Vigoda. Entropy decay in the Swendsen Wang dynamics on $\mathbb{Z}^d$. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1551–1564, 2021. doi:10.1145/3406325.3451095.

4 Antonio Blanca, Pietro Caputo, Alistair Sinclair, and Eric Vigoda. Spatial mixing and non-local markov chains. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1965–1980, 2018. doi:10.1137/1.9781611975031.128.

5 Antonio Blanca, Zongchen Chen, Daniel Štefankovič, and Eric Vigoda. The Swendsen–Wang dynamics on trees. *Random Structures & Algorithms*, 62(4):791–831, 2022. doi:10.1002/RSA.21121.

6 Antonio Blanca, Zongchen Chen, and Eric Vigoda. Swendsen-Wang dynamics for general graphs in the tree uniqueness region. *Random Structures & Algorithms*, 56(2):373–400, 2020. doi:10.1002/rsa.20858.

7 Antonio Blanca and Reza Gheissari. Random-cluster dynamics on random regular graphs in tree uniqueness. *Communications in Mathematical Physics*, 386(2):1243–1287, 2021. doi:10.1007/s00220-021-04093-z.

8 Antonio Blanca and Reza Gheissari. Sampling from Potts on random graphs of unbounded degree via random-cluster dynamics. *Annals of Applied Probability*, 33(6B):4997–5049, 2023. Extended abstract appeared in the Proceedings of RANDOM 2022.

9 Antonio Blanca and Alistair Sinclair. Random-Cluster dynamics in $\mathbb{Z}^2$. *Probability Theory and Related Fields*, 168:821–847, 2017. Extended abstract appeared in the Proceedings of SODA 2016. doi:10.1007/s00440-016-0725-1.

10 Antonio Blanca and Zhezheng Song. Cutoff for the Swendsen-Wang dynamics on the complete graph. *arXiv preprint arXiv:2507.20482*, 2025.

11 Antonio Blanca and Xusheng Zhang. Rapid Mixing of Global Markov Chains via Spectral Independence: The Unbounded Degree Case. In *Proceedings of the 27th International Conference on Randomization and Computation (RANDOM)*, page 53, 2023.

12 Charles Carlson, Ewan Davies, and Alexandra Kolla. Efficient algorithms for the Potts model on small-set expanders. *Chicago Journal of Theoretical Computer Science*, 1:1–31, 2024.

13 Charlie Carlson, Ewan Davies, Nicolas Fraiman, Alexandra Kolla, Aditya Potukuchi, and Corrine Yap. Algorithms for the ferromagnetic Potts model on expanders. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 344–355, 2022. doi:10.1109/FOCS54457.2022.00040.

14 Amin Coja-Oghlan, Andreas Galanis, Leslie Ann Goldberg, Jean Bernoulli Ravelomanana, Daniel Štefankovič, and Eric Vigoda. Metastability of the Potts ferromagnet on random

regular graphs. *Communications in Mathematical Physics*, 401:185–225, 2023. doi:10.1007/s00220-023-04644-6.

15 Andrea Collevecchio, Timothy M Garoni, Timothy Hyndman, and Daniel Tokarev. The worm process for the ising model is rapidly mixing. *Journal of Statistical Physics*, 164:1082–1102, 2016.

16 Colin Cooper and Alan M. Frieze. Mixing properties of the Swendsen–Wang process on classes of graphs. *Random Structures and Algorithms*, 15:242–261, 1999. doi:10.1002/(SICI)1098-2418(199910/12)15:3/4\%3C242::AID-RSA4\%3E3.0.CO;2-C.

17 Paul Cuff, Jian Ding, Oren Louidor, Eyal Lubetzky, Yuval Peres, and Allan Sly. Glauber dynamics for the mean-field Potts model. *Journal of Statistical Physics*, 149:432–477, 2012.

18 Robert G. Edwards and Alan D. Sokal. Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm. *Physical Review. D. Particles and Fields. Third Series*, 38(6):2009–2012, 1988. doi:10.1103/PhysRevD.38.2009.

19 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Swendsen-Wang algorithm on the mean-field Potts model. *Random Structures & Algorithms*, 54(1):82–147, 2019. doi:10.1002/RSA.20768.

20 Andreas Galanis, Daniel Štefankovič, Eric Vigoda, and Linji Yang. Ferromagnetic Potts model: Refined #BIS-hardness and related results. *SIAM Journal on Computing*, 45(6):2004–2065, 2016. doi:10.1137/140997580.

21 Reza Gheissari, Eyal Lubetzky, and Yuval Peres. Exponentially slow mixing in the mean-field Swendsen–Wang dynamics. *Annales de l'Institut Henri Poincare (B)*, 56(1):68–86, 2019. Extended abstract appeared in SODA 2018.

22 Leslie Ann Goldberg and Mark Jerrum. Approximating the partition function of the ferromagnetic Potts model. *Journal of the ACM*, 59(5):1–31, 2012. doi:10.1145/2371656.2371660.

23 Vivek K. Gore and Mark R. Jerrum. The Swendsen-Wang process does not always mix rapidly. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 674–681, 1997. doi:10.1145/258533.258662.

24 Heng Guo and Mark Jerrum. Random cluster dynamics for the Ising model is rapidly mixing. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1818–1827, 2017. doi:10.1137/1.9781611974782.118.

25 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic Pirogov–Sinai theory. *Probability Theory and Related Fields*, 176(3):851–895, 2020. Extended abstract appeared in the Proceedings of STOC 2019. doi:10.1007/s00440-019-00928-y.

26 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding*, pages 409–426, 1994.

27 Mark Huber. Perfect sampling using bounding chains. *The Annals of Applied Probability*, 14(2):734–753, 2004.

28 Svante Janson and Malwina J Luczak. Susceptibility in subcritical random graphs. *Journal of Mathematical Physics*, 49(12), 2008.

29 Matthew Jenssen, Peter Keevash, and Will Perkins. Algorithms for #BIS-hard problems on expander graphs. *SIAM Journal on Computing*, 49(4):681–710, 2020. doi:10.1137/19M1286669.

30 M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22:1087–1116, 1993. doi:10.1137/0222066.

31 Heejune Kim. Cutoff phenomenon of the glauber dynamics for the Ising model on complete multipartite graphs in the high temperature regime. *Markov Processes And Related Fields*, 28(1):113–148, 2022.

32 David A Levin, Malwina J Luczak, and Yuval Peres. Glauber dynamics for the mean-field Ising model: cut-off, critical power law, and metastability. *Probability Theory and Related Fields*, 146:223–265, 2010.

33 David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Society, 2017.

34 Yun Long, Asaf Nachmias, Weiyang Ning, and Yuval Peres. A power law of order 1/4 for critical mean field Swendsen-Wang dynamics. *Memoirs of the American Mathematical Society*, 232(1092), 2014. doi:10.1090/memo/1092.

35 Eyal Lubetzky and Allan Sly. Cutoff for the Ising model on the lattice. *Inventiones mathematicae*, 191:719–755, 2013.

36 Eyal Lubetzky and Allan Sly. Universality of cutoff for the Ising model. *Annals of Probability*, 45(6):3664–3696, 2017.

37 Malwina Luczak and Tomasz Luczak. The phase transition in the cluster-scaled model of a random graph. *Random Structures & Algorithms*, 28(2):215–246, 2006. doi:10.1002/RSA.20088.

38 Danny Nam and Allan Sly. Cutoff for the Swendsen-Wang dynamics on the lattice. *Annals of Probability*, 47(6):3705–3761, 2019. doi:10.1214/19-AOP1344.

39 Dana Randall and David Wilson. Sampling spin configurations of an Ising system. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 959–960, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314945.

40 Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in Monte Carlo simulations. *Physical Review Letters*, 58:86–88, 1987. doi:10.1103/PhysRevLett.58.86.

41 Mario Ullrich. Swendsen–Wang is faster than single-bond dynamics. *SIAM Journal on Discrete Mathematics*, 28(1):37–48, 2014. doi:10.1137/120864003.

42 Remco van der Hofstad, Takashi Kumagai, et al. Glauber dynamics for Ising models on random regular graphs: cut-off and metastability. *Alea: Latin American journal of probability and mathematical statistics*, 18(1):1441–1482, 2021.

A Mixing time lower bound

We will use the following result for proving the the lower bound on the mixing time as stated in Theorem 1.

▶ **Lemma 21.** *If $X \sim \mu$, then $\|\alpha(X) - m\|_\infty \leq A/\sqrt{n}$ with probability at least 3/4 for a suitable constant $A > 0$.*

Similar results are available in the literature, but we were not able to find this particular variant, so, for completeness, we provide a proof of Lemma 21 in the full version of this paper [10].

Proof of Theorem 1 (Lower bound). By Lemma 21, if $X \sim \mu$, then $\|\alpha(X) - m\|_\infty \leq A/\sqrt{n}$ with probability at least 3/4 for a suitable constant A. Now, take X_0 to be a configuration where all vertices are assigned the same spin. By Lemmas 16 and 17, there exists γ such that after $T = c(\beta, q) \log n + \gamma$ steps, with probability $\geq 3/4$, we have $\|\alpha(X_T) - m\|_\infty > \frac{A}{\sqrt{n}}$. Hence, $\|X_T - \mu\|_{\text{TV}} > 1/4$ and thus that the SW dynamics has not mixed after T steps. ◀

B Proof of Lemma of 15

The idea behind this proof is that we can show that once A1 and A2 are achieved, they are preserved for a constant number of steps with constant probability, providing more attempts for the coupling from Lemma 14 to succeed at least once.

To formalize this, let $\mathcal{P}(r)$ be the set of pairs of configurations of the projection chain A and A' such that any full configurations X and X' that project onto A or A', respectively, satisfy: $\|\alpha(X) - m\|_\infty \leq r/\sqrt{n}$, $\|\alpha(Y) - m\|_\infty \leq r/\sqrt{n}$, $\max_{k \in [q]} |\alpha_{1,k}(X) - \alpha_{1,k}(Y)| \leq r/\sqrt{n}$.

▶ **Fact 22.** *For $\delta \in (0, 1)$, there exists $r = r(\delta) > 0$ such that $\Pr(Z_{t+1} \in \mathcal{P}(r) \,|\, Z_t \in \mathcal{P}(r)) \geq \delta$.*

Proof. Let X_t and Y_t be two configurations that project onto $Z_t = Z \in \mathcal{P}(r)$. Then, $\|\alpha(X_t) - m\|_\infty \le r/\sqrt{n}$. Let us assume that $\alpha_1(X_t) \ge a$ (the case when $\alpha_1(X_t) < a$ is analogous). By Fact 8, $F(\alpha_1(X_t)) \le F(a + r/\sqrt{n})$. Moreover, by Fact 7 and the mean value theorem, there exists a constant $\eta \in (0,1)$ such that $F(a+r/\sqrt{n}) - a = \eta r/\sqrt{n}$ which implies that $(a + r/\sqrt{n}) - F(a + r/\sqrt{n}) = (1-\eta)r/\sqrt{n}$. In order for $\|\alpha(X_{t+1}) - m\|_\infty \ge r/\sqrt{n}$, we would need $\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \ge \frac{(1-\eta)r}{\sqrt{n}}$. From Lemma 18, for any large enough constant r, we have

$$\Pr\left(\|\alpha(X_{t+1}) - \kappa(F(\alpha_1(X_t)))\|_\infty \ge \frac{(1-\eta)r}{\sqrt{n}}\right) \le O(r^{-2}).$$

The same holds for Y_{t+1}.

Similarly, by assumption, $d_t^{(k)} = |\alpha_1(X_t) - \alpha_{1,k}(X_t)| \le r/\sqrt{n}$. Then, using Lemma 19, for any large enough constant r, we have that there exists $\hat{\eta} \in (0,1)$ such that

$$\Pr\left(d_{t+1}^{(k)} \ge \frac{r}{\sqrt{n}}\right) \le \Pr\left(|d_{t+1}^{(k)} - \hat{\eta} d_t^{(k)}| \ge \frac{(1-\hat{\eta})r}{\sqrt{n}}\right) \le O(r^{-2}).$$

The same holds for all $k \in [q]$ by a union bound, and for Y_{t+1}. The result then follows by another union bound over the four events. ◀

Proof of Lemma 15. We consider the coupling that in each step aims to couple two instances of the projection chain using the coupling from Lemma 14. If it succeeds, the two instances remain coupled in all future steps. For ease of notation, let $\mathcal{P}$ be the set of pairs of configurations of the projection chain that satisfy A1 and A2 for a constant $\hat{C}$ sufficiently large, let $\mathcal{S}_t$ be the event that $A_t = A'_t$, and let $Z_t = (A_t, A'_t)$. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}$ be the subset of $\mathcal{P}$ that contains pairs of distinct configurations: i.e., the pairs $(A, A') \in \mathcal{P}$ such that $A \ne A'$.

It suffices to show that for a suitable constant ℓ, we have $\Pr\left(\cap_{i=1}^{\ell} \neg\mathcal{S}_{t+i}\right) < 1/4$. For this, we obtain first the following recurrence:

$$\begin{aligned}
&\Pr\left(\cap_{i=1}^{\ell} \neg\mathcal{S}_{t+i}\right) = \Pr\left(\neg\mathcal{S}_{t+\ell} \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \\
&\le \sum_{Z \in \mathcal{P}} \Pr\left(\neg\mathcal{S}_{t+\ell}, Z_{t+\ell-1} = Z \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) + \Pr(Z_{t+\ell-1} \notin \mathcal{P}) \\
&= \sum_{Z \in \hat{\mathcal{P}}} \Pr\left(\neg\mathcal{S}_{t+\ell}, Z_{t+\ell-1} = Z \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) + \Pr(Z_{t+\ell-1} \notin \mathcal{P}), \qquad (24)
\end{aligned}$$

where the first inequality follows from the law of total probability, and the second equality from the fact that under our coupling the probability of $\neg\mathcal{S}_{t+\ell}$ is 0 when $Z \in \mathcal{P} \setminus \hat{\mathcal{P}}$. The sum in right-hand-side of (24) satisfies:

$$\begin{aligned}
&= \sum_{Z \in \hat{\mathcal{P}}} \Pr\left(\neg\mathcal{S}_{t+\ell} \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}, Z_{t+\ell-1} = Z\right) \Pr(Z_{t+\ell-1} = Z \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \\
&= \sum_{Z \in \hat{\mathcal{P}}} \Pr\left(\neg\mathcal{S}_{t+\ell} \mid Z_{t+\ell-1} = Z\right) \Pr(Z_{t+\ell-1} = Z \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \\
&\le (1-\delta) \sum_{Z \in \hat{\mathcal{P}}} \Pr(Z_{t+\ell-1} = Z \mid \cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right) \\
&\le (1-\delta) \Pr\left(\cap_{i=1}^{\ell-1} \neg\mathcal{S}_{t+i}\right),
\end{aligned}$$

where the second equality follows from the Markov property; in particular, note that $\{Z_t\}$ is a Markov chain and $Z \in \hat{\mathcal{P}}$ is such that $\neg\mathcal{S}_{t+\ell-1}$ holds and can be dropped from the conditioning. The first inequality follows from Lemma 14.

Observe also that for $j \in \{1, \ldots, \ell - 1\}$, it follows from Fact 22 that there is a constant $\delta_2 > 0$ that we can choose arbitrarily close to 1 by taking $\hat{C}$ large enough so that

$$\begin{aligned}\Pr(Z_{t+j} \in \mathcal{P}) &\geq \sum_{Z \in \mathcal{P}} \Pr(Z_{t+j} \in \mathcal{P} \mid Z_{t+j-1} = Z) \Pr(Z_{t+j-1} = Z) \\ &\geq \sum_{Z \in \mathcal{P}} \delta_2 \Pr(Z_{t+j-1} = Z) \geq \delta_2 \Pr(Z_{t+j-1} \in \mathcal{P}),\end{aligned}$$

and iterating we get $\Pr(Z_{t+j} \in \mathcal{P}) \geq \delta_2^j$, since by assumption $Z_t \in \mathcal{P}$. Then,

$$\Pr\left(\cap_{i=1}^{\ell} \neg \mathcal{S}_{t+i}\right) \leq (1-\delta) \Pr\left(\cap_{i=1}^{\ell-1} \neg \mathcal{S}_{t+i}\right) + 1 - \delta_2^{\ell-1}.$$

Iterating once again,

$$\Pr\left(\cap_{i=1}^{\ell} \neg \mathcal{S}_{t+i}\right) \leq (1-\delta)^{\ell} + \sum_{j=1}^{\ell-1} (1-\delta_2^j)(1-\delta)^{\ell-j-1} \leq (1-\delta)^{\ell} + (1-\delta)^{\ell} \sum_{j=1}^{\ell-1} \frac{1-\delta_2^j}{(1-\delta)^{j+1}}.$$

Now first fixing ℓ so that $(1-\delta)^{\ell} < 1/8$, we can then pick δ_2 close enough to 1 so that $\sum_{j=1}^{\ell-1} \frac{1-\delta_2^j}{(1-\delta)^{j+1}} < 1$, and the result follows. ◀

Iterating Non-Aggregative Structure Compositions

Marius Bozga
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Radu Iosif
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Florian Zuleger
Institute of Logic and Computation, Technische Universität Wien, Austria

Abstract

An aggregative composition is a binary operation obeying the principle that the whole is determined by the sum of its parts. The development of graph algebras, on which the theory of formal graph languages is built, relies on aggregative compositions that behave like disjoint union, except for a set of well-marked interface vertices from both sides, that are joined. The same style of composition has been considered in the context of relational structures, that generalize graphs and use constant symbols to label the interface.

In this paper, we study a non-aggregative composition operation, called *fusion*, that joins non-deterministically chosen elements from disjoint structures. The sets of structures obtained by iteratively applying fusion do not always have bounded tree-width, even when starting from a tree-width bounded set. First, we prove that the problem of the existence of a bound on the tree-width of the closure of a given set under fusion is decidable, when the input set is described inductively by a finite *hyperedge-replacement* (HR) grammar, written using the operations of aggregative composition, forgetting and renaming of constants. Such sets are usually called *context-free*. Second, assuming that the closure under fusion of a context-free set has bounded tree-width, we show that it is the language of an effectively constructible HR grammar. A possible application of the latter result is the possiblity of checking whether all structures from a non-aggregatively closed set having bounded tree-width satisfy a given monadic second order logic formula.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Grammars and context-free languages

Keywords and phrases Hyperedge replacement, Tree-width

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.18

Related Version *Full Version*: https://arxiv.org/abs/2510.06019

Funding Marius Bozga and Radu Iosif wish to acknowledge the support of the French National Research Agency project Non-Aggregative Resource COmpositions (NARCO) under grant number ANR-21-CE48-0011. Florian Zuleger wishes to acknowledge the support of the FWF project AUTOSARD: "Automated Sublinear Amortised Resource Analysis of Data Structures" No. P36623 and the project VASSAL: "Verification and Analysis for Safety and Security of Applications in Life" funded by the European Union under Horizon Europe WIDERA Coordination and Support Action/Grant Agreement No. 101160022.

1 Introduction

The *tree-width* of a graph is a numerical measure of how "tree-like" the graph is. This notion extends naturally to the relational structures, used to define the semantics of classical first and second-order logic. Relational structures generalize a broad range of graph-like objects, such as edge-labeled graphs, hypergraphs, and multi-edge graphs. Tree-width plays a foundational role in logic and verification. Courcelle's theorem [6] shows that *Monadic Second-order Logic* (MSO) is decidable on classes of structures having bounded tree-width,

© Marius Bozga, Radu Iosif, and Florian Zuleger;
licensed under Creative Commons License CC-BY 4.0

while Seese's theorem [19] asserts that unbounded tree-width leads to undecidability of MSO theories. Thus, proving that a given class of structures has bounded tree-width is tantamount for establishing the decidability of logical theories for that class.

In principle, one is interested in reasoning about infinite families of structures. These families are typically generated inductively from a finite set of basic building blocks, using operations such as composition and renaming. These operations are usually formalized by the Hyperedge Replacement (HR) algebra introduced by Courcelle [8]. The principle of inductive definition is captured by the notion of a *context-free* set, i.e., the language of a finite grammar written using HR operations, or equivalently, the set of evaluations of a set of ground HR terms recognized by a tree automaton.

For both graphs and relational structures, HR algebras are built on *aggregative composition* operations, in which the whole is determined by the sum of its parts. These compositions are typically defined as the disjoint union of the arguments, where the elements designated by shared constants on both sides are joined together. In other words, aggregative composition preserves the identity of the substructures while merging them at known interface points. Importantly, context-free sets of graphs and relational structures defined by grammars based on such bounded-interface[1] aggregative compositions have bounded tree-width.

In this paper, we investigate a more flexible but less controlled operation, called *non-aggregative fusion*. Fusion allows elements from two disjoint structures to be merged non-deterministically, even when they are not marked by constants. To maintain a certain level of semantic coherence, we require fusion to be constrained by a *coloring* discipline: elements can only be joined if their colors (i.e., sets of designated unary relations) are disjoint. This idea stems from existing work in the area of reasoning about the correctness of systems with dynamically reconfigurable connectivity, such as distributed protocols [1] or pointer structures with aliasing [15].

Due to its nondeterministic nature, fusion can cause a dramatic shift in structure: even if a set **S** consists of structures having bounded tree-width, by taking the closure of **S** under fusion one may introduce infinitely many structures of unbounded tree-width. This phenomenon raises two natural and fundamental questions:

1. Given a context-free set of relational structures, does the closure of this set under fusion have bounded tree-width ?
2. If the answer to the above question is yes, is this closure again a context-free set ?

The result this paper is that both questions have a positive answer (Theorem 9):

1. The existence of a bound on the tree-width of the closure by fusion of a context-free set is a decidable problem.
2. If the fusion-closure of context-free set has bounded tree-width, then it is the language of an effectively constructible context-free grammar, that uses only aggregative composition.

These results provide tools for reasoning about nondeterministic structural iteration. For instance, one can check MSO properties over the tree-width bounded fusion-closure of a context-free set, thereby extending algorithmic verification techniques to a broader class of systems. We sketch below two possible application domains that have motivated our work.

Separation Logic of Relations (SLR). A key motivation for studying non-aggregative fusion comes from SLR, a generalization of classical Separation Logic to relational structures. This logic has been first considered for relational databases and object-oriented languages [14].

[1] Vertex-replacement algebras [8] using disjoint union and edge addition between arbitrarily large interfaces may produce sets of unbounded tree-width.

More recently, SLR (combined with inductive definitions [12]) has been proposed as an assertion language for the verification of distributed reconfigurable systems [1]. Here, the *separating conjunction* $\phi * \psi$ means that the models of two formulæ ϕ and ψ must not have overlapping interpretations of the same relation symbol.

A subtle but crucial point is that, while the separating conjunction enforces *disjointness* of the tuples that interpret a relation symbol, i.e., that tuples cannot overlap in *all* positions, the tuples may overlap in *some* positions. However, by the disjointness of $*$, such overlapping is only possible if the variables at these positions do not occur within the same unary relation symbol [2]. From a semantic point of view, this behavior corresponds precisely to our notion of fusion: joining elements of two separate structures is allowed, as long as their sets of unary relation labels are disjoint. Thus, fusion abstracts the semantics of SLR, where aliasing is controlled implicitly by colors (i.e., sets of relation symbols) and the semantics of the separating conjunction. Moreover, considering inductive definitions on top of the basic SLR logic is akin to considering context-free sets generated by recursive grammars here.

Chemical and Biological Systems. We believe that fusion-like operations naturally arise in the modeling of chemical and biological systems, where complex structures (e.g., proteins or polymeric carbon chains) are formed by joining smaller components through local interactions. Here non-aggregative fusion abstracts the joining of components not only at fixed attachment points, but also through general, property-based interactions, modeled via color compatibility in our framework. Studying the tree-width of these structures is essential for enabling automated reasoning about their properties. We consider the exploration of this area as future work.

Related Work

The notion of composition is central to substructural logics [17]. One of the foremost such logics is Bunched Implications (BI), whose first definition of semantics is based on partially ordered monoids (i.e., the multiplicative connective is interpreted as the multiplication in the monoid) [16]. In particular, the monoidal semantics of BI (and many other follow-up logics) do not assume the composition to be aggregative. The advent of the more popular semantics of BI based on finite partial functions, called *heaps*, has made aggregative composition popular among the users of Separation Logic [13, 18]. We list below several substructural logics where composition is not aggregative.

Docherty and Pym developped Intuitionistic Layered Graph Logic (ILGL), a substructural logic tailored to reasoning about graph structures with a fixed, non-commutative and non-associative notion of layering [9]. Calcagno et al. [2] introduce Context Logic as a framework for local reasoning about structured data, emphasizing compositionality through structural connectives that describe data and context separately rather than flattening them into aggregates. In [3], they formalize these connectives as modal operators and demonstrate that such non-aggregative reasoning is essential for expressing weakest preconditions and verifying updates. Cardelli et al. [4] present a spatial logic for reasoning about graphs that, like our work, emphasizes local, non-aggregative composition via structural connectives such as spatial conjunction.

[2] For each unary relation symbol r in the alphabet, the SLR formula $\mathsf{r}(x) * \mathsf{r}(y)$ entails $x \neq y$.

2 Definitions

Given integers i and j, we write $[i..j]$ for the set $\{i, i+1, \ldots, j\}$, assumed to be empty if $i > j$. For a set A, we denote by $\mathrm{pow}(A)$ its powerset. By writing $B \subseteq_{fin} A$ we mean that B is a finite subset of A. The cardinality of a finite (multi)set A is written $\mathrm{card}(A)$. By writing $A = A_1 \uplus A_2$ we mean that A_1 and A_2 partition A, i.e., $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$. The n-times Cartesian product of A with itself is denoted A^n and the set of possibly empty (resp. nonempty) sequences of elements from A by A^* (resp. A^+). Multisets are denoted as $\{\!\{a, b, \ldots\}\!\}$, $\sqcup$ and $\sqcap$ denote the operations of multiset union and intersection, respectively. The multi-powerset (i.e., the set of multisets) of A is written $\mathrm{mpow}(A)$.

2.1 Relational Structures

Let $\mathbb{R}$ be a finite alphabet of *relation symbols* $\mathsf{r} \in \mathbb{R}$, of arities $\#\mathsf{r} \geq 1$, and $\mathbb{C}$ be a countably infinite set of *constants* $\mathsf{c} \in \mathbb{C}$ of arity zero. As usual, relation symbols of arity 1, 2 and 3 are called unary, binary and ternary, respectively. A *$\mathcal{C}$-structure*, for some finite set $\mathcal{C} \subseteq_{fin} \mathbb{C}$ of constants, is a pair $\mathsf{S} = (\mathsf{U_S}, \sigma_\mathsf{S})$, where $\mathsf{U_S}$ is a finite set called *universe* and σ_S is an *interpretation* that maps each relation symbol $\mathsf{r} \in \mathbb{R}$ to a subset of $\mathsf{U_S}^{\#\mathsf{r}}$ and each constant $\mathsf{c} \in \mathcal{C}$ to an element of $\mathsf{U_S}$. The *sort* of the structure S is the set $\mathcal{C}$. Two structures are *disjoint* iff their universes are disjoint and *isomorphic* iff they are defined over the same alphabet and differ only by a renaming of their elements[3]. For a given sort $\mathcal{C} \subseteq_{fin} \mathbb{C}$, we denote by $\mathcal{S}(\mathcal{C})$ the set of $\mathcal{C}$-structures.

We define the *composition* of two relational structures as the component-wise union of disjoint isomorphic copies of the structures followed by joining the elements that interpret the common constants. This is the same as the *gluing* operation defined by Courcelle [7, Definition 2.1], that we recall below, for self-completeness:

▶ **Definition 1.** *Let $\mathsf{S} \in \mathcal{S}(\mathcal{C})$ be a structure and $\sim \subseteq \mathsf{U_S} \times \mathsf{U_S}$ be an equivalence relation, where $[u]_\sim$ denotes the equivalence class of $u \in \mathsf{U_S}$. The* quotient *of S with respect to $\sim$ is the $\mathcal{C}$-structure $\mathsf{S}_{/\sim}$ defined as follows:*

$$\begin{aligned}
\mathsf{U}_{\mathsf{S}_{/\sim}} &\stackrel{\text{def}}{=} \{[u]_\sim \mid u \in \mathsf{U_S}\} \\
\sigma_{\mathsf{S}_{/\sim}}(\mathsf{r}) &\stackrel{\text{def}}{=} \{([u_1]_\sim, \ldots, [u_{\#\mathsf{r}}]_\sim) \mid (u_1, \ldots, u_{\#\mathsf{r}}) \in \sigma_\mathsf{S}(\mathsf{r})\}, \text{ for each } \mathsf{r} \in \mathbb{R} \\
\sigma_{\mathsf{S}_{/\sim}}(\mathsf{c}) &\stackrel{\text{def}}{=} [\sigma_\mathsf{S}(\mathsf{c})]_\sim, \text{ for each } \mathsf{c} \in \mathcal{C}
\end{aligned}$$

Let $\mathsf{S}_i = (\mathsf{U}_i, \sigma_i)$ be disjoint $\mathcal{C}_i$-structures, for $i = 1, 2$, and $\approx \subseteq (\mathsf{U}_1 \uplus \mathsf{U}_2) \times (\mathsf{U}_1 \uplus \mathsf{U}_2)$ be the least equivalence relation such that $\sigma_1(\mathsf{c}) \approx \sigma_2(\mathsf{c})$, for all $\mathsf{c} \in \mathcal{C}_1 \cap \mathcal{C}_2$. The composition of S_1 with S_2 is the $\mathcal{C}_1 \cup \mathcal{C}_2$-structure $\mathsf{S}_1 * \mathsf{S}_2 = (\mathsf{U}, \sigma)$, where:

$$\begin{aligned}
\mathsf{U} &\stackrel{\text{def}}{=} \{[u]_\approx \mid u \in \mathsf{U}_1 \uplus \mathsf{U}_2\} \\
\sigma(\mathsf{r}) &\stackrel{\text{def}}{=} \{([u_1]_\approx, \ldots, [u_{\#\mathsf{r}}]_\approx) \mid (u_1, \ldots, u_{\#\mathsf{r}}) \in \sigma_1(\mathsf{r}) \uplus \sigma_2(\mathsf{r})\}, \text{ for each } \mathsf{r} \in \mathbb{R} \\
\sigma(\mathsf{c}) &\stackrel{\text{def}}{=} [\sigma_i(\mathsf{c})]_\approx, \text{ if } \mathsf{c} \in \mathcal{C}_i, \text{ for each } \mathsf{c} \in \mathcal{C}_1 \cup \mathcal{C}_2 \quad (\text{i.e., } [\sigma_1(\mathsf{c})]_\approx = [\sigma_2(\mathsf{c})]_\approx \text{ if } \mathsf{c} \in \mathcal{C}_1 \cap \mathcal{C}_2)
\end{aligned}$$

We remark that the composition of $\emptyset$-structures S_1 and S_2 is the same as their disjoint union, denoted $\mathsf{S}_1 \uplus \mathsf{S}_2$.

[3] See, e.g., [10, Section A3] for a formal definition of isomorphism between structures.

The composition of structures is *aggregative*, meaning that it keeps both structures separate except for the interpretation of the common constants. In the following, we define a non-aggregative *fusion* operation that matches also some of the elements which are not interpretations of common constants. In contrast to the deterministic composition, the equivalence relation that matches elements in the fusion operation is chosen nondeterministically.

Before formalizing the notion of non-aggregative fusion, we introduce a generic mechanism for controlling which pairs of elements are allowed to join. We assume a designated set of unary relation symbols $\mathfrak{C} \subseteq \mathbb{R}$. The sets of relation symbols $\gamma \in \mathrm{pow}(\mathfrak{C})$ are called *colors*. Given some $\mathcal{C}$-structure $\mathsf{S} = (\mathsf{U}_\mathsf{S}, \sigma_\mathsf{S})$, we denote by $\mathsf{col}_\mathsf{S}(u) = \{\mathsf{r} \in \mathfrak{C} \mid u \in \sigma_\mathsf{S}(\mathsf{r})\}$ the color of each element $u \in \mathsf{U}_\mathsf{S}$. Note that the empty set is a color.

Back to the definition of non-aggregative fusion, we use colors to prevent joining elements labeled with non-disjoint colors. This is captured by the following notion of compatibility:

▶ **Definition 2.** *Let $\mathsf{S} = (\mathsf{U}, \sigma)$ be a $\mathcal{C}$-structure. A relation $\sim \subseteq \mathsf{U} \times \mathsf{U}$ is* compatible *with S if and only if $\mathsf{col}_\mathsf{S}(u_1) \cap \mathsf{col}_\mathsf{S}(u_2) = \emptyset$, for each pair $u_1 \sim u_2$.*

We now define the non-aggregative fusion operation. The operation is non-deterministic, i.e., returns a (possibly empty) set of structures. For reasons of simplicity, the fusion operation is only defined for structures of sort $\emptyset$, i.e., for structures that do not interpret any constants. This restriction can be lifted at the expense of complexifying the definition below, by considering fusion in which the interpretation of common constants in both structures must always be joined, as in Definition 1.

Given disjoint sets A and B, a relation $\sim \subseteq A \times B$ is an *A-B matching* iff $\{a, b\} \cap \{a', b'\} = \emptyset$, for all distinct pairs $(a, b), (a', b') \in \sim$. The least equivalence relation that contains $\sim$ is denoted $\equiv_\sim \subseteq (A \uplus B) \times (A \uplus B)$. We say that an equivalence relation $\equiv_\sim$ is *k-generated* iff $\sim$ is a matching consisting of k pairs.

▶ **Definition 3.** *Let $\mathsf{S}_i = (\mathsf{U}_i, \sigma_i)$, for $i = 1, 2$, be two disjoint $\emptyset$-structures. The* fusion *of S_1 and S_2 is the following set of $\emptyset$-structures:*

$$\mathsf{F}(\mathsf{S}_1, \mathsf{S}_2) \stackrel{\text{def}}{=} \{(\mathsf{S}_1 * \mathsf{S}_2)_{/\equiv_\sim} \mid \sim \text{ non-empty } \mathsf{U}_1\text{-}\mathsf{U}_2 \text{ matching compatible with } \mathsf{S}_1 * \mathsf{S}_2\}$$

Let $\mathbf{S}$ be a set of $\emptyset$-structures. The closure of $\mathbf{S}$ under fusion is the least set $\mathsf{F}^*(\mathbf{S})$ such that $\mathbf{S} \cup \{\mathsf{F}(\mathsf{S}_1, \mathsf{S}_2) \mid \mathsf{S}_1, \mathsf{S}_2 \in \mathsf{F}^*(\mathbf{S})\} \subseteq \mathsf{F}^*(\mathbf{S})$.

Note that $\mathsf{F}(\mathsf{S}_1, \mathsf{S}_2) = \emptyset$ iff $\mathsf{col}_{\mathsf{S}_1}(u_1) \cap \mathsf{col}_{\mathsf{S}_1}(u_2) \neq \emptyset$, for all pairs $(u_1, u_2) \in \mathsf{U}_1 \times \mathsf{U}_2$. The problems considered in the rest of this paper concern the uses of fusion, in addition to the composition and the unary operations on structures introduced next.

2.2 An Algebra of Structures

We recall the definitions of sorted terms and algebras [6, Definition 1.1]. Let Σ be a countably infinite set of *sorts* and let $\mathbb{F}$ be a countably infinite set of *function symbols*, where the set $\mathbb{F}$ is called a *signature*. Each $f \in \mathbb{F}$ has an associated tuple of argument sorts $\alpha(f)$ and a value sort $\rho(f)$. The arity of f, denoted $\#f$, is the length of $\alpha(f)$. Moreover, each variable has a sort. A $\mathbb{F}$-term $t[x_1, \ldots, x_n]$ is built as usual from function symbols and variables $x_1, \ldots, x_n$ of matching sorts. A ground term is a term without variables. A trivial term consists of a single variable. A term t' is a subterm of t iff there exists a term $u[x]$ such that $t = u[t']$, where $u[t']$ denotes the replacement of x by t' in u. The sort of a term t, denoted $\rho(t)$ is the value sort of the top-most symbol, i.e., either the value sort $\rho(f)$ of the top-most function symbol f, in case of a non-trivial term t, or the sort $\rho(x)$ of the variable x, in case of a trivial

term x. A position p in a term t is a node of the tree that uniquely represents t, in the usual way (see [5] for a formal definition). $\mathcal{T}(\mathcal{F})$ denotes the set of ground terms having function symbols taken from a finite set $\mathcal{F} \subseteq_{fin} \mathbb{F}$.

An $\mathbb{F}$-*algebra* $\mathcal{A} = (\{\mathsf{A}_s\}_{s\in\Sigma}, \{f^{\mathcal{A}}\}_{f\in\mathbb{F}})$ consists of domains A_s of each sort $s \in \Sigma$ and interprets the function symbols $f \in \mathbb{F}$ as functions $f^{\mathcal{A}} : \mathsf{A}_{s_1} \times \ldots \times \mathsf{A}_{s_n} \to \mathsf{A}_{\rho(f)}$, where $\alpha(f) = (s_1, \ldots, s_n)$. By the domain of $\mathcal{A}$ we understand the set $\mathsf{A} = \bigcup_{s\in\Sigma} \mathsf{A}_s$. We denote by $t^{\mathcal{A}}$ the interpretation of an $\mathbb{F}$-term t in $\mathcal{A}$, i.e., the function obtained by replacing each function symbol that occurs in t by its interpretation. In particular, $t^{\mathcal{A}}$ is an element of the domain of $\mathcal{A}$ if t is ground.

We define an algebra of structures, called $\mathcal{HR}$, with sorts $\Sigma_{\mathcal{HR}} \stackrel{\text{def}}{=} \{\mathcal{C} \mid \mathcal{C} \subseteq_{fin} \mathbb{C}\}$, where each universe $\mathsf{HR}_{\mathcal{C}}$ is the set of $\mathcal{C}$-structures. The signature $\mathbb{F}_{\mathcal{HR}}$ consists of:

- constant symbols $\mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$, for $\mathsf{r} \in \mathbb{R}$ and $\mathcal{C}_i \subseteq_{fin} \mathbb{C}$ such that either $\mathcal{C}_i = \mathcal{C}_j$ or $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all $1 \leq i < j \leq \#\mathsf{r}$, interpreted as $\mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})^{\mathcal{HR}} \stackrel{\text{def}}{=} (\mathsf{U}, \sigma)$, where:

$$\mathsf{U} \stackrel{\text{def}}{=} \{u_1, \ldots, u_{\#\mathsf{r}}\} \text{ for some (possibly equal) elements } u_1, \ldots, u_{\#\mathsf{r}} \text{ such that } u_i = u_j \iff \mathcal{C}_i = \mathcal{C}_j, \text{ for all } 1 \leq i < j \leq \#\mathsf{r}$$
$$\sigma(\mathsf{r}) \stackrel{\text{def}}{=} \{(u_1, \ldots, u_{\#\mathsf{r}})\} \text{ and } \sigma(\mathsf{r}') \stackrel{\text{def}}{=} \emptyset, \text{ for all } \mathsf{r}' \in \mathbb{R} \setminus \{\mathsf{r}\}$$
$$\sigma(\mathsf{c}) \stackrel{\text{def}}{=} u_i, \text{ for all } \mathsf{c} \in \mathcal{C}_i \text{ and } 1 \leq i \leq \#\mathsf{r}$$

- binary function symbols $\oplus_{\mathcal{C},\mathcal{C}'}$, for $\mathcal{C}, \mathcal{C}' \subseteq_{fin} \mathbb{C}$, interpreted by the composition operation $*$ from Definition 1 applied to structures of sorts $\mathcal{C}$ and $\mathcal{C}'$, respectively, see Figure 1 (a).
- unary function symbols $\mathsf{rename}_{\mathcal{C}}^{\alpha}$, for all $\mathcal{C}, \mathcal{C}' \subseteq_{fin} \mathbb{C}$ and surjective function $\alpha : \mathcal{C} \to \mathcal{C}'$, interpreted as the operations $\mathsf{rename}_{\mathcal{C}}^{\alpha\,\mathcal{HR}} : \mathsf{HR}_{\mathcal{C}} \to \mathsf{HR}_{\mathcal{C}'}$ where, for each $\mathcal{C}$-structure S, the output $\mathsf{S}' = \mathsf{rename}_{\mathcal{C}}^{\alpha}(\mathsf{S})$ is defined below:

$$\mathsf{U}_{\mathsf{S}'} \stackrel{\text{def}}{=} \mathsf{U}_{\mathsf{S}} \quad \sigma_{\mathsf{S}'}(\mathsf{r}) \stackrel{\text{def}}{=} \sigma_{\mathsf{S}}(\mathsf{r}) \quad \sigma_{\mathsf{S}'}(\alpha(\mathsf{c})) \stackrel{\text{def}}{=} \sigma_{\mathsf{S}}(\mathsf{c}), \text{ for all } \mathsf{r} \in \mathbb{R} \text{ and } \mathsf{c} \in \mathcal{C} \setminus \mathcal{C}'$$

- unary function symbols $\mathsf{forget}_{\mathcal{C}}^{\mathcal{C}'}$, for $\mathcal{C} \subseteq_{fin} \mathbb{C}$ and $\mathcal{C}' \subseteq \mathcal{C}$, interpreted as the operations $\mathsf{forget}_{\mathcal{C}}^{\mathcal{C}'\,\mathcal{HR}} : \mathsf{HR}_{\mathcal{C}} \to \mathsf{HR}_{\mathcal{C}\setminus\mathcal{C}'}$ where, for each $\mathcal{C}$-structure S, the output $\mathsf{S}' = \mathsf{forget}_{\mathcal{C}}^{\mathcal{C}'\,\mathcal{HR}}(\mathsf{S})$ is defined below:

$$\mathsf{U}_{\mathsf{S}'} \stackrel{\text{def}}{=} \mathsf{U}_{\mathsf{S}} \quad \sigma_{\mathsf{S}'}(\mathsf{r}) \stackrel{\text{def}}{=} \sigma_{\mathsf{S}}(\mathsf{r}) \quad \sigma_{\mathsf{S}'}(\mathsf{c}) \stackrel{\text{def}}{=} \sigma_{\mathsf{S}}(\mathsf{c}), \text{ for all } \mathsf{r} \in \mathbb{R} \text{ and } \mathsf{c} \in \mathcal{C}$$

To ease the notation, we omit the sorts of the arguments from the $\mathcal{HR}$ function symbols $\mathbb{F}_{\mathcal{HR}}$ when they are understood from the context.

▶ **Example 4.** The two leftmost graphs in Figure 1 (a) are the values of the following terms, respectively (singleton sets are denoted by their elements, to avoid clutter):

$$t_1 \stackrel{\text{def}}{=} \mathsf{rename}^{\mathsf{c}_3\to\mathsf{c}_1}\,(\mathsf{forget}^{\mathsf{c}_1}\,(a(\mathsf{c}_0, \mathsf{c}_1, \mathsf{c}_2) \oplus c(\mathsf{c}_1, \mathsf{c}_3)) \oplus b(\mathsf{c}_0, \mathsf{c}_3))$$
$$t_2 \stackrel{\text{def}}{=} b(\mathsf{c}_0, \mathsf{c}_1) \oplus \mathsf{forget}^{\mathsf{c}_2}\,(a(\mathsf{c}_1, \mathsf{c}_3, \mathsf{c}_2))$$

The graph in Figure 1 (c) is the value of the term $t_0 \stackrel{\text{def}}{=} \mathsf{rename}^{\mathsf{c}_0\leftrightarrow\mathsf{c}_1}\,(\mathsf{forget}^{\mathsf{c}_2}(t_1 \oplus t_2))$.

We comment on the relationship to the algebra of relational structures introduced in [7, Definition 2.3], as a generalization of both the hyperedge-replacement (HR) algebra of hypergraphs and the vertex-replacement (VR) algebra of binary graphs, to relational structures. As a matter of fact, we do not use this algebra. Instead, our algebra of relational structures algebra $\mathcal{HR}$ follows the standard HR algebra on hypergraphs [6]. This relationship can be easily understood by viewing relational structures as hypergraphs in the

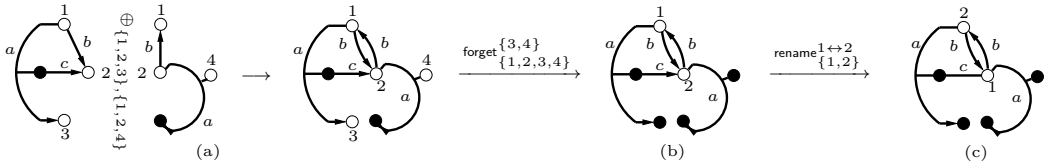


Figure 1 HR operations on structures over the alphabet $\{a, b, c\}$, where $\#a = 3$ and $\#b = \#c = 2$. The order of vertices attached to an edge is indicated by an arrow pointing to the last vertex.

adjacency encoding, i.e., each tuple $(u_1, \ldots, u_{\#\mathsf{r}})$ from the interpretation of a relation symbol r corresponds to a r-labeled hyperedge attached to the vertices $u_1, \ldots, u_{\#\mathsf{r}}$ in this order. For reasons of space, we omit further details.

We now introduce the subalgebras of $\mathcal{HR}$ that define the tree-width parameter of a structure. For this, we assume some enumeration of the constants $\mathbb{C} = \{\mathsf{c}_0, \mathsf{c}_1, \ldots\}$ and denote by $\mathbb{F}_{\mathcal{HR}^{\leq k}}$ the subset of $\mathbb{F}_{\mathcal{HR}}$ consisting of the function symbols whose argument and value sorts are all contained in $\{\mathsf{c}_0, \ldots, \mathsf{c}_k\}$.

▶ **Definition 5.** *The* tree-width *of a structure* S*, denoted* $\mathrm{tw}(\mathsf{S})$*, is the minimal integer* $k \geq 0$ *for which there exists a ground term* $t \in \mathcal{T}(\mathbb{F}_{\mathcal{HR}^{\leq k}})$ *such that* $t^{\mathcal{HR}} = \mathsf{S}$*. A set* $\mathbf{S}$ *of structures has* bounded tree-width *if and only if the set* $\{\mathrm{tw}(\mathsf{S}) \mid \mathsf{S} \in \mathbf{S}\}$ *is finite.*

In particular, for each tree-width bounded set $\mathbf{S}$, there exists a set $\mathcal{T}$ of ground terms and a finite set $\mathcal{C} \subseteq_{fin} \mathbb{C}$ of constants such that each term $t \in \mathcal{T}$ uses only constants from $\mathcal{C}$ and $\mathbf{S} = \{t^{\mathcal{HR}} \mid t \in \mathcal{T}\}$.

This algebraic definition of tree-width of a relational structure is analogous to the definition of the tree-width of a hypergraph using a subalgebra of HR defined by a restriction of sorts to finite sets of vertex labels (see, e.g., [8, Proposition 1.19] for a proof of equivalence between the graph-theoretic and algebraic definitions of tree-width for hypergraphs). Moreover, the tree-width of a structure can be equivalently defined in terms of the tree-width of its Gaifman-graph:

▶ **Definition 6.** *Let* S *be a* $\mathcal{C}$*-structure. The* Gaifman graph *of* S *is the simple undirected graph* $\mathsf{Gaif}(\mathsf{S}) = (V, E)$*, where* $V \stackrel{\text{def}}{=} \mathsf{U}_\mathsf{S}$ *and* $E \stackrel{\text{def}}{=} \{\{u_i, u_j\} \mid (u_1, \ldots, u_{\#\mathsf{r}}) \in \sigma_\mathsf{S}(\mathsf{r}), 1 \leq i \neq j \leq \#\mathsf{r}, \mathsf{r} \in \mathbb{R}\}$.

It is known that the tree-width of a structure equals the tree-width of its Gaifman graph [11, Proposition 11.27].

We recall below two standard notions of graph theory. Given binary graphs G and H, we say that H is a *minor* of G iff H is obtained from a subgraph of G by *edge contractions*, where the contraction of a binary edge $e \in E_G$ attached to vertices u and v means deleting e and joining u and v into a single vertex x (the edges attached to x are the ones attached to either u or v). It is well-known that the tree-width of each minor of a graph G is bounded by the tree-width of G.

A *$n \times m$-grid* is a binary graph whose vertices can be labeled with pairs $(i, j) \in [1..n] \times [1..m]$ such that there is an edge between (i, j) and (i', j') iff either $i < n$, $i' = i + 1$ and $j' = j$ or $j < m$, $i' = i$ and $j' = j + 1$. It is well-known[4] that each $n \times n$-grid has tree-width n. By bluring the distinction between isomorphic grids, we obtain the following:

[4] This can be shown by using the characterisation of tree-width in terms of the cops and robber game [20].

▶ **Proposition 7.** *A set of structures whose Gaifman graphs contain infinitely many non-isomorphic square grids has unbounded tree-width.*

2.3 Context-Free Sets of Structures

Context-free sets are usually defined as languages of grammars, i.e., finite sets of inductive rules written using nonterminals and function symbols from a given signature. To simplify some of the upcoming proofs, we use here an equivalent definition of contex-free sets based on recognisable sets of ground terms, defined using tree automata [5]. For self-containment reasons, we briefly introduce context-free grammars and discuss their equivalence with tree automata in Section 3.5 (see the statement of Theorem 24).

Let $\mathcal{F} \subseteq_{fin} \mathbb{F}$ be a finite signature of function symbols. A *tree automaton* over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, F, \rightarrow)$, where Q is a finite set of states, $F \subseteq Q$ is a set of accepting states and $\rightarrow$ is a set of transition rules of the form $(q_1, \ldots, q_{\#f}) \xrightarrow{f} q$, where $q_1, \ldots, q_{\#f}, q \in Q$ and $f \in \mathcal{F}$. A run π of $\mathcal{A}$ over a ground term $t \in \mathcal{T}(\mathcal{F})$ maps each position p within t to a state $q = \pi(p)$ if the automaton has a rule $(\pi(p_1), \ldots, \pi(p_{\#f})) \xrightarrow{f} q$, where $p_1, \ldots, p_{\#f}$ are the positions of the children of p in t. A ground term t is accepted by $\mathcal{A}$ iff $\mathcal{A}$ has a run that labels the root of t with an accepting state. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of ground terms accepted by $\mathcal{A}$. A set $T \subseteq \mathcal{T}(\mathcal{F})$ is *recognisable* iff it is the language of a tree automaton over the finite signature $\mathcal{F}$.

▶ **Definition 8.** *A set* $\mathbf{S}$ *of structures is* context-free *if and only if there exists a recognisable set of ground terms* $T \subseteq \mathcal{T}(\mathcal{F})$*, over a finite signature* $\mathcal{F} \subseteq \mathbb{F}_{\mathcal{HR}}$*, such that* $\mathbf{S} = \{t^{\mathcal{HR}} \mid t \in T\}$.

Note that a context-free set of structures has finitely many sorts, because the signature of terms used to describe the set is finite. For this reason, any context-free set of structures has bounded tree-width. On the other hand, there are bounded tree-width sets which are not context-free, for instance the set of linear structures over the alphabet $\{a, b, c\}$ of binary relations of the form $a^n b^n c^n$, for all $n \geq 1$.

3 The Closure of Context-Free Sets under Fusion

This section is concerned with the statement and proof of the main result of the paper (Theorem 9). For simplicity, we assume that $\mathbf{S}$ is a set of *connected* structures, where connectivity of a structure S means that there exists an undirected path in $\mathsf{Gaif}(\mathsf{S})$ between each pair of elements $u_1, u_2 \in \mathsf{U}_\mathsf{S}$. We note that, because $\mathbf{S}$ contains only connected structures, each structure from its closure $\mathsf{F}^*(\mathbf{S})$ is necessarily connected.

The assumption of $\mathbf{S}$ being a context-free set of connected structures loses no generality, because it is possible, from a tree automaton $\mathcal{A}$ such that $\mathbf{S} = \mathcal{L}(A)^{\mathcal{HR}}$, to build a tree automaton $\mathcal{B}$ such that $\mathcal{L}(B)^{\mathcal{HR}}$ is the set of connected substructures of a structure in $\mathbf{S}$. Intuitively, $\mathcal{B}$ is obtained from $\mathcal{A}$ be labeling each state q with finite information concerning the existence of a path between each pair of constant symbols in each structure that is the value of a ground term recognized by q in $\mathcal{A}$. This construction can be used to generalize the statement of Theorem 9 below from connected to arbitrary structures. We will detail this construction in an extended version of the present article.

▶ **Theorem 9.** *Let* $\mathbf{S}$ *be a context-free set of connected* $\emptyset$*-structures.*

1. $\mathsf{F}^*(\mathbf{S})$ *has bounded tree-width if and only if* $\mathsf{F}^*(\mathbf{S})$ *is context-free.*
2. *It is decidable whether* $\mathsf{F}^*(\mathbf{S})$ *has bounded tree-width.*

An obvious consequence of this theorem is the decidability of the problem: *given a context-free set* $\mathbf{S}$*, is* $\mathsf{F}^*(\mathbf{S})$ *context-free?*

We give an overview of the proof before going into technical details. The core idea is the equivalence between the (A) tree-width boundedness of the closure $\mathsf{F}^*(\mathbf{S})$ of a context-free set $\mathbf{S}$ and (B) the non-existence of two structures $\mathsf{S}_1, \mathsf{S}_2 \in \mathsf{F}^*(\mathbf{S})$ having each at least three elements each $u_i, v_i, w_i \in \mathsf{U}_{\mathsf{S}_i}$, labeled with disjoint colors γ_i, for $i = 1, 2$. This equivalence is established via a third, more technical, condition about the disjointness relations between the colors that may occur in a structure from $\mathsf{F}^*(\mathbf{S})$. The latter implies that the matching relation of each fusion of two structures from $\mathsf{F}^*(\mathbf{S})$ is generated by at most two pairs of elements with compatible colors.

The equivalence between (A) and (B) is used to prove both points of Theorem 9. For point (1) suppose, for a contradiction, that (B) does not hold. Then, S_1 and S_2 can be composed by joining their u_i, v_i or w_i elements, for $i = 1, 2$, respectively, as in Figure 2. Consequently, the set of Gaifman graphs corresponding to the structures in $\mathsf{F}^*(\mathbf{S})$ contains an infinite set of grid minors, thus $\mathsf{F}^*(\mathbf{S})$ has unbounded tree-width (Proposition 7). Else, if (B) holds (i.e., such structures cannot be found), we prove that the matching relation considered in the fusion of any two structures is generated by either one or two pairs of elements. In each of these cases, by adding a finite number of constants to the signature of the $\mathbb{F}_{\mathcal{HR}}$-terms from the language of $\mathcal{A}$, we can build a tree automaton $\mathcal{A}^*$ such that $\mathcal{L}(\mathcal{A}^*)^{\mathcal{HR}} = \mathsf{F}^*(\mathbf{S})$, thus taking care of point (1) of the theorem.

To prove point (2), we rely on the equivalence of (A) and (B) and show that (B) is decidable. This is done by arguing that the existence of two structures with the above property is equivalent to the existence of two multiset abstractions of structures $\{\!\{\gamma_1, \gamma_1, \gamma_1\}\!\}$, $\{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\}$ in the domain of multisets of colors having multiplicity at most three. These multiset abstractions of colors can be effectively computed by a finite fixpoint iteration over the rules of the tree automaton that recognizes the set of ground terms which evaluates to the elements of $\mathbf{S}$.

3.1 Color Multisets

In the following, let $\mathbf{S}$ be a context-free set of $\emptyset$-structures. We denote the set of colors by $\Gamma \stackrel{\text{def}}{=} \mathrm{pow}(\mathfrak{C})$, where $\mathfrak{C}$ is a fixed finite set of unary relation symbols. First, we define an abstraction of structures as finite multisets of colors:

▶ **Definition 10.** *The* multiset color abstraction $\mathsf{S}^\sharp \in \mathrm{mpow}(\Gamma)$ *of a structure* S *is* $\mathsf{S}^\sharp \stackrel{\text{def}}{=} \{\!\{\mathsf{col}_{\mathsf{S}}(u) \mid u \in \mathsf{U}_{\mathsf{S}}\}\!\}$. *For an integer* $k \geq 0$, *the* k-multiset color abstraction $\mathsf{S}^{\sharp k} \subseteq \mathrm{mpow}(\Gamma)$ *is* $\mathsf{S}^{\sharp k} \stackrel{\text{def}}{=} \{M \subseteq \mathsf{S}^\sharp \mid \mathrm{card}(M) \leq k\}$. *These abstractions are lifted to sets of structures as sets of multisets* $\mathbf{S}^\sharp \stackrel{\text{def}}{=} \{\mathsf{S}^\sharp \mid \mathsf{S} \in \mathbf{S}\}$ *and* $\mathbf{S}^{\sharp k} \stackrel{\text{def}}{=} \bigcup_{\mathsf{S} \in \mathbf{S}} \mathsf{S}^{\sharp k}$.

Note that $\mathsf{S}^\sharp$ is a multiset, whereas $\mathsf{S}^{\sharp k}$ is a set of multisets. When lifted to sets of structures, both $\mathbf{S}^\sharp$ and $\mathbf{S}^{\sharp k}$ are sets of multisets.

The core of the proof of Theorem 9 is the equivalence between the following two conditions stated formally below:

$$\mathrm{tw}(\mathsf{F}^*(\mathbf{S})) \leq k, \text{ for some } k \geq 1 \tag{A}$$

$$\{\!\{\gamma_1, \gamma_1, \gamma_1\}\!\}, \{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\} \in (\mathsf{F}^*(\mathbf{S}))^{\sharp 3} \Rightarrow \gamma_1 \cap \gamma_2 \neq \emptyset, \text{ for all } \gamma_1, \gamma_2 \in \Gamma \tag{B}$$

Note that condition (A) is more general than the premiss of Theorem 9. We prove the (A) $\Rightarrow$ (B) direction below.

▶ **Lemma 11.** *If* $\mathbf{S}$ *has bounded tree-width, then (A) implies (B).*

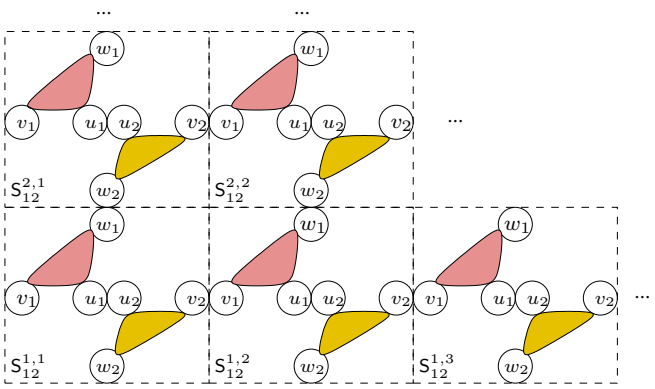


Figure 2 Building structures whose Gaifman graphs have infinitely large grid minors.

Proof. By contradiction, assume that there exist $\{\!\{\gamma_1, \gamma_1, \gamma_1\}\!\}, \{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\} \in (\mathsf{F}^*(\mathbf{S}))^{\sharp 3}$ such that $\gamma_1 \cap \gamma_2 = \emptyset$. Then, there exist structures $\mathsf{S}_1, \mathsf{S}_2 \in \mathsf{F}^*(\mathbf{S})$ such that $\{\!\{\gamma_1, \gamma_1, \gamma_1\}\!\} \in \mathsf{S}_1^\sharp$ and $\{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\} \in \mathsf{S}_2^\sharp$. We shall use S_1 and S_2 to build infinitely many structures whose Gaifman graphs have arbitrarily large square grid minors, as illustrated in Figure 2.

First, construct the structure $\mathsf{S}_{12} \in \mathsf{F}^*(\mathbf{S})$ by fusing one pair (u_1, u_2), having colors γ_1 and γ_2, respectively. Let v_1 and w_1 (resp. v_2 and w_2) be the remaining distinct elements of S_{12}, having color γ_1 (resp. γ_2) from S_1 and S_2, respectively. For an arbitrarily large integer $n \geq 1$, consider $n \times n$ disjoint copies $(\mathsf{S}_{12}^{i,j})_{i,j=1,n}$ of S_{12}. Let $\approx^{1,j}$ be the equivalence relation generated by $\{(v_1^{1,j}, v_2^{1,j-1})\}$ and $\approx^{i,1}$ be generated by $\{(w_2^{i,1}, w_1^{i-1,1})\}$, $\approx^{i,j}$ be generated by $\{(v_1^{i,j}, v_2^{i,j-1}), (w_2^{i,j}, w_1^{i-1,j})\}$, for all $i, j = 2, n$. Second, construct the grid-like connected structure $X^{n,n} \in \mathsf{F}^*(\mathbf{S})$:

$$X^{n,n} = (...(...((\mathsf{S}_{12}^{1,1} * \mathsf{S}_{12}^{1,2})_{/\approx^{1,2}} * \mathsf{S}_{12}^{2,1})_{/\approx^{2,1}} * ... * \mathsf{S}_{12}^{i,j})_{/\approx^{i,j}} * ... * \mathsf{S}_{12}^{n,n})_{/\approx^{n,n}}$$

where structures $\mathsf{S}_{12}^{i,j}$ are added to the fusion in the increasing order of $i + j$. We can show that $\mathsf{Gaif}(X^{n,n})$ has an $n \times n$ square grid minor. Finally, as n can be taken arbitrarily large, we conclude that $\mathsf{F}^*(\mathbf{S})$ does not have bounded tree-width, which contradicts (A). ◀

3.2 Color Schemes

For the proof of the (A) "⇐" (B) direction, we first organize the set of colors using the *RGB color schemes* defined below:

▶ **Definition 12.** *A partition* $(\Gamma^{red}, \Gamma^{green}, \Gamma^{blue})$ *of* Γ *is an* RGB color scheme *if and only if:*

1. $\gamma_1 \cap \gamma_2 \neq \emptyset$, *for all* $\gamma_1, \gamma_2 \in \Gamma^{blue}$,
2. $\gamma_1 \cap \gamma_2 \neq \emptyset$, *for all* $\gamma_1 \in \Gamma^{green}$ *and all* $\gamma_2 \in \Gamma^{blue}$,
3. *for all* $\gamma_1 \in \Gamma^{red}$ *there exists* $\gamma_2 \in \Gamma^{blue}$ *such that* $\gamma_1 \cap \gamma_2 = \emptyset$.

Note that an RGB color scheme is fully specified by the set Γ^{blue}. Indeed, any color not in Γ^{blue} is unambiguously placed within Γ^{red} or Γ^{green}, depending on whether or not it is disjoint from some color in Γ^{blue}. In particular, if $\Gamma^{blue} = \emptyset$ then $\Gamma^{red} = \emptyset$ and $\Gamma^{green} = \Gamma$. For example, Figure 3 shows several RGB color schemes for the set $\mathfrak{C} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ of unary relation symbols.

Because a fusion operation only joins element with disjoint colors, blue elements can only be joined with red elements, green elements can be joined with green or red elements, whereas red elements can be joined with elements of any other color. We define below what is meant for a set of structures to conform to an RGB color scheme:

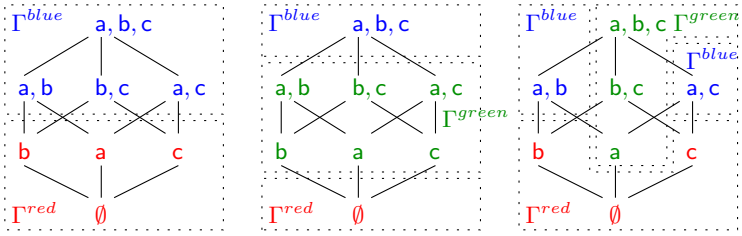


Figure 3 Examples of RGB color schemes.

Table 1 The types of structures obtained by fusion, where $\emptyset$ means that the result of the fusion is the empty set.

$\mathsf{F}(\mathsf{S}_1, \mathsf{S}_2)$	S_2 of R type	S_2 of G type	S_2 of B type
S_1 of R type	R, G, B	G, B	B
S_1 of G type	G, B	G, B	$\emptyset$
S_1 of B type	B	$\emptyset$	$\emptyset$

▶ **Definition 13.** *A set* $\mathbf{S}$ *of structures* conforms to $(\Gamma^{red}, \Gamma^{green}, \Gamma^{blue})$ *if and only if:*

1. *for all structures* $\mathsf{S} \in \mathbf{S}$*, if* $\mathsf{col}_\mathsf{S}(u) \in \Gamma^{red}$*, for some element* $u \in \mathsf{U}_\mathsf{S}$*, then* $\mathsf{col}_\mathsf{S}(u') \in \Gamma^{blue}$*, for all other elements* $u' \in \mathsf{U}_\mathsf{S} \setminus \{u\}$*, and*
2. $\mathsf{S}^\sharp \cap \Gamma^{green} \subseteq \{\!\{\gamma, \gamma \mid \gamma \in \Gamma^{green}\}\!\}$*, for all structures* $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$.

In other words, $\mathbf{S}$ conforms to a given color scheme if each structure from $\mathbf{S}$ has either a single red and the rest blue, or at most occurrences of the same green color and the rest blue elements. Moreover, the number of occurrences of a green color must not exceed two, for each structure obtained by taking fusions of some structures in $\mathbf{S}$. This observation justifies the following notion of *type* of a structure:

▶ **Definition 14.** *A structure* S *is of* type R *if it has exactly one red element and the rest blue,* G *if it has at least one green element and the rest blue and* B *if it has only blue elements.*

Note that there can be structures of neither R, G or B type, but these are the only types of interest, as justified by the following:

▶ **Lemma 15.** *Let* $\mathbf{S}$ *be a set of structures conforming to an RGB color scheme. Then, each structure* $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$ *is of type either* R*,* G *or* B.

Proof. By induction on the construction of $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$ from one or more structures from $\mathbf{S}$. Table 1 summarizes the possible types of $\mathsf{F}(\mathsf{S}_1, \mathsf{S}_2)$ on structures S_1 and S_2 of types R, G or B, respectively. ◀

The (B) "⇒" (A) direction will be established via a third condition (C), which is conformance to an RGB color scheme defined by taking the Γ^{blue} set to be the colors occurring three times in some structure from $\mathsf{F}^*(\mathbf{S})$:

▶ **Lemma 16.** *If (B) holds then:*

$$\mathbf{S} \text{ conforms to } (\Gamma^{red}, \Gamma^{green}, \Gamma^{blue}), \text{ where } \Gamma^{blue} \stackrel{\text{def}}{=} \{\gamma \in \Gamma \mid \{\!\{\gamma, \gamma, \gamma\}\!\} \in (\mathsf{F}^*(\mathbf{S}))^{\sharp 3}\} \qquad \text{(C)}$$

Proof. We show that $\mathbf{S}$ conforms to the $(\Gamma^{red}, \Gamma^{green}, \Gamma^{blue})$ RGB color scheme from the statement, by checking the two points of Definition 13:

(1) Let $\mathsf{S} \in \mathbf{S}$ and prove that for any two colors $\gamma_1, \gamma_2 \in \mathfrak{C}$, if $\{\!\{\gamma_1, \gamma_2\}\!\} \subseteq \mathsf{S}^\sharp$ and $\gamma_1 \in \Gamma^{red}$ then $\gamma_2 \in \Gamma^{blue}$. Since $\gamma_1 \in \Gamma^{red}$, there must exists a color $\gamma_1' \in \Gamma^{blue}$, such that $\gamma_1 \cap \gamma_1' = \emptyset$, by Definition 12. By the definition of Γ^{blue}, this further implies $\{\!\{\gamma_1', \gamma_1', \gamma_1'\}\!\} \in (\mathsf{F}^*(\mathbf{S}))^{\sharp 3}$. Henceforth, there exists a structure $\mathsf{S}' \in \mathsf{F}^*(\mathbf{S})$ such that $\{\!\{\gamma_1', \gamma_1', \gamma_1'\}\!\} \subseteq \mathsf{S}'^\sharp$. We can now use S' and three disjoint copies of S to build a new structure S'' by gluing progressively, each one of the three elements of color γ_1' in S' to the element of color γ_1 of S. Then, by construction, the structure S'' will also contain three elements of color γ_2, one from each disjoint copy of S. Therefore, $\{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\} \in \mathsf{S}''^\sharp$ and because $\mathsf{S}'' \in \mathsf{F}^*(\mathbf{S})$ this implies $\{\!\{\gamma_2, \gamma_2, \gamma_2\}\!\} \in (\mathsf{F}^*(\mathbf{S}))^{\sharp 3}$ and therefore $\gamma_2 \in \Gamma^{blue}$.

(2) By contradiction, let $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$ be such that $\mathsf{S}^\sharp \sqcap \Gamma^{green} \not\subseteq \{\!\{\gamma, \gamma \mid \gamma \in \Gamma^{green}\}\!\}$. Then there exists $\gamma' \in (\mathsf{S}^\sharp \sqcap \Gamma^{green}) \setminus \{\!\{\gamma, \gamma \mid \gamma \in \Gamma^{green}\}\!\}$, i.e., $\gamma' \in \Gamma^{green}$ and $\{\!\{\gamma', \gamma', \gamma'\}\!\} \subseteq \mathsf{S}^\sharp$. The latter implies $\{\!\{\gamma', \gamma', \gamma'\}\!\} \in \mathsf{S}^{\sharp 3} \subseteq (\mathsf{F}^*(\mathbf{S}))^{\sharp 3}$. But this implies $\gamma' \in \Gamma^{blue}$ according to the definition of the RGB color scheme, contradicting $\gamma' \in \Gamma^{green}$. ◀

In the rest of this and the next subsections, we are concerned with the proof of the following implication, that establishes the equivalence of (A) and (B). As previously mentioned, this direction of the proof uses the third condition (C) that is, conformance to the RGB color scheme from the statement of Lemma 16:

▶ **Lemma 17.** *If* $\mathbf{S}$ *has bounded tree-width and (C) holds then (A) holds.*

The proof of the above lemma is split into two technical results (Lemmas 18 and 19). The first (Lemma 18) involves reasoning about the number of pairs of elements that are joined by the fusion operation in order to obtain a structure from $\mathsf{F}^*(\mathbf{S})$:

▶ **Lemma 18.** *If* $\mathbf{S}$ *conforms to an RGB scheme* $(\Gamma^{red}, \Gamma^{green}, \Gamma^{blue})$ *and* $\mathsf{S} = (\mathsf{S}_1 \uplus \mathsf{S}_2)_{/\approx}$ *for some* $\mathsf{S}_i = (\mathsf{U}_i, \sigma_i) \in \mathsf{F}^*(\mathbf{S})$*, for* $i = 1, 2$*, and some equivalence closure* $\approx$ *of a* U_1-U_2 *matching, then exactly one of the following holds:*

1. $\approx$ *is* 1*-generated, or*
2. $\approx$ *is* 2*-generated,* S *is of type* B*, and either:*
   a. S_1*,* S_2 *are both of type* R*, or*
   b. S_1*,* S_2 *are both of type* G *and* $\mathrm{card}(\mathsf{S}_1{}^\sharp \sqcap \Gamma^{green}) = \mathrm{card}(\mathsf{S}_2{}^\sharp \sqcap \Gamma^{green}) = 2$.

Proof. We distinguish two cases:

- S_1 is of type R: If S_2 is of type B or G then S_1 and S_2 can be fused only by equivalences $\approx$ generated by a single pair, that contains the element from the support of S_1 with color in Γ^{red}, thus matching the case (1) from the statement. Else, if S_2 is of type R then S_1 and S_2 can be fused by equivalences generated by at most two pairs, each containing an element with color from Γ^{red}, from either S_1 or S_2, thus matching the case 2a from the statement. In this latter case, S is of type B because joining a red with a blue element always results in a blue element.
- S_1, S_2 are both of type G: By contradiction, assume they can be fused by an equivalence $\approx$ generated by three pairs of elements $(u_{1i}, u_{2i})_{i=1,2,3}$. Let $G_{1i} = \mathsf{col}_{\mathsf{S}_1}(u_{1i})$, $G_{2i} = \mathsf{col}_{\mathsf{S}_2}(u_{2i})$ be the colors from Γ^{green} of the matching elements in the two structures, for $i = 1, 2, 3$. Then, we can construct structures using S_1 and S_2 where any of these colors repeat strictly more than twice, henceforth, contradicting the conformance property to the RGB color scheme. The principle of the construction is depicted in Figure 4. Finally, note that the construction depicted in Figure 4 fuse actually only pairs of colors (G_{1i}, G_{2i}) for $i = 1, 2$. Henceforth, the conformance property is also contradicted if S_1 and S_2 can be fused by a 2-generated equivalence relation $\approx$, such that the support of either S_1 or S_2 contains more than three elements with colors in Γ^{green}. By the same argument, it follows that S is of type B, if $\approx$ is generated by two pairs of green elements. ◀

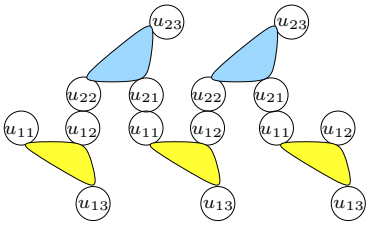


Figure 4 Fusion of G structures by 3-generated matchings.

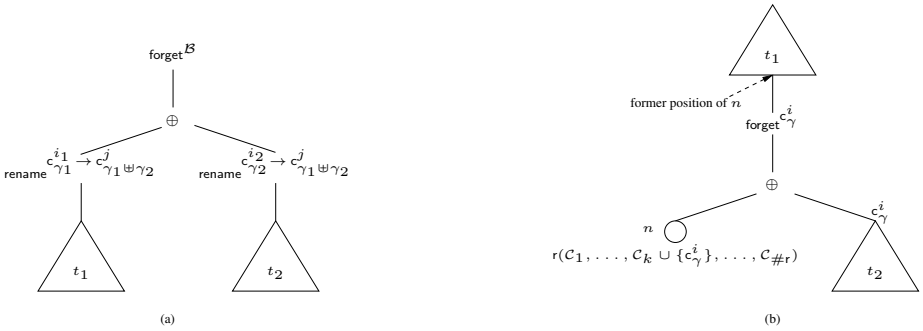


Figure 5 The $\mathsf{join}(t_1, t_2, \mathsf{c}^{i_1}_{\gamma_1}, \mathsf{c}^{i_2}_{\gamma_2})$ (a) and $\mathsf{append}(t_1, t_2, n, k, \mathsf{c}^i_\gamma)$ (b) operations.

3.3 Fusions as Operations on Terms

The second technical result required for the proof of Lemma 17 is a characterization of the k-generated fusion, for $k = 1, 2$, via operations on witness $\mathbb{F}_{\mathcal{HR}}$-terms. We assume that $\mathbf{S}$ is a given tree-width bounded set of structures that conforms to a fixed RGB color scheme $(\Gamma^{red}, \Gamma^{green}, \Gamma^{blue})$. The goal is to prove that $\mathrm{tw}(\mathsf{F}^*(\mathbf{S})) \leq \mathrm{tw}(\mathbf{S}) + K$, for an integer $K \geq 0$.

Since $\mathbf{S}$ has bounded tree-width, there exists a set $\mathcal{T}$ of terms such that $\mathbf{S} = \{t^{\mathcal{HR}} \mid t \in \mathcal{T}\}$ and a finite set $\mathcal{C}$ of constants such that each term $t \in \mathcal{T}$ uses only constants from $\mathcal{C}$ and assume w.l.o.g. $\mathcal{C}$ to be the least such set. Moreover, consider the following set of *special* constants $\overline{\mathcal{C}} \stackrel{\text{def}}{=} \{\mathsf{c}^i_\gamma \mid \gamma \in \Gamma,\ 1 \leq i \leq 2\}$, with the following intuition. Recall that each element of a structure $t^{\mathcal{HR}}$ is the common interpretation of all the constants $\mathsf{c} \in \mathcal{C}_i$ in the label $\mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$ of a leaf of t. By adding c^i_γ to the set $\mathcal{C}_i$, we mean that the color of that element in the structure is γ. A constant is *visible* in a term if it is not in the scope of a rename or forget operation. Let $K \stackrel{\text{def}}{=} \mathrm{card}(\overline{\mathcal{C}})$ and note that $\mathrm{card}(K) = 2 \cdot \mathrm{card}(\Gamma)$.

We shall prove that $\mathrm{tw}(\mathsf{S}) \leq \mathrm{tw}(\mathbf{S}) + K$ by building, for any $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$, a term t that uses only constants from $\mathcal{C} \uplus \overline{\mathcal{C}}$, such that $\mathsf{forget}^{\overline{c}}\left(t^{\mathcal{HR}}\right) = \mathsf{S}$. We then note that S has tree-width at most $\mathrm{card}(\mathcal{C}) + K$ and obtain $\mathrm{tw}(\mathsf{S}) \leq \mathrm{card}(\mathcal{C}) + K = \mathrm{tw}(\mathbf{S}) + K$. Since the choice of $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$ was arbitrary, this leads to $\mathrm{tw}(\mathsf{F}^*(\mathbf{S})) \leq \mathrm{tw}(\mathbf{S}) + K$.

In the following, we understand terms as trees whose nodes are labeled by function symbols from $\mathbb{F}_{\mathcal{HR}}$. For each node n of a term t, we write $\mathsf{lab}(n)$ for its label. The children of each node n form an ordered sequence of length equal to the arity of $\mathsf{lab}(n)$. In order to the build the witness terms for the structures $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$, we make use of the following operations on terms t, t_1 and t_2 having constants in $\mathcal{C} \uplus \overline{\mathcal{C}}$:

1. $\mathsf{label}(t, n, k, \mathsf{c}^i_\gamma)$, where n is a leaf of t, $\mathsf{lab}(n) = \mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$ and $k \in [1..\#\mathsf{r}]$: Let $1 \leq k_1 < \ldots < k_\ell \leq \#\mathsf{r}$ be the indices of the sets of constants from $\mathsf{lab}(n)$ that are equal to $\mathcal{C}_k$ (see the definition of $\mathcal{HR}$ in Section 2.2). The operation changes the label of n in t only, by replacing the set $\mathcal{C}_{k_j}$ with $\mathcal{C}_{k_j} \cup \{\mathsf{c}^i_\gamma\}$, for each $1 \leq j \leq \ell$.

2. $\mathsf{join}(t_1, t_2, \mathsf{c}^{i_1}_{\gamma_1}, \mathsf{c}^{i_2}_{\gamma_2})$: The result is the following term, for a nondeterministic choice of j, such that $\mathsf{c}^{j}_{\gamma_1 \uplus \gamma_2}$ is not visible in either t_1 or t_2 (the operation is undefined otherwise):

$$\mathsf{forget}^{\mathcal{B}}(\mathsf{rename}^{\mathsf{c}^{i_1}_{\gamma_1} \to \mathsf{c}^{j}_{\gamma_1 \uplus \gamma_2}}(t_1) \ \oplus \mathsf{rename}^{\mathsf{c}^{i_2}_{\gamma_2} \to \mathsf{c}^{j}_{\gamma_1 \uplus \gamma_2}}(t_2)), \ \mathcal{B} \stackrel{\text{def}}{=} \{\mathsf{c}^{j}_{\gamma_1 \uplus \gamma_2} \mid \gamma_1 \uplus \gamma_2 \in \Gamma^{blue}\}$$

We refer to Figure 5 (a) for an illustration. In addition, we consider an overloaded version of $\mathsf{join}(t_1, t_2, \mathsf{c}^{i_{11}}_{\gamma_{11}}, \mathsf{c}^{i_{12}}_{\gamma_{12}}, \mathsf{c}^{i_{21}}_{\gamma_{21}}, \mathsf{c}^{i_{22}}_{\gamma_{22}})$ that fuses the interpretation of $\mathsf{c}^{i_{1j}}_{\gamma_{1j}}$ with that of $\mathsf{c}^{i_{2j}}_{\gamma_{2j}}$, for both $j = 1, 2$. This definition is similar to the one above, thus omitted for brevity.

3. $\mathsf{append}(t_1, t_2, n, k, \mathsf{c}^{i}_{\gamma})$, where n is a leaf of t_1, $\mathsf{lab}(n) = \mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$ and $k \in [1..\#\mathsf{r}]$: the result is the term $t_1[n/\mathsf{forget}^{\mathsf{c}^{i}_{\gamma}}(\mathsf{label}(n, n, k, \mathsf{c}^{i}_{\gamma}) \oplus t_2)]$, where $t[n/s]$ denotes the substitution of the leaf n by the term s in t. We refer to Figure 5 (b) for an illustration.

Then, Lemma 17 is an immediate consequence of the following lemma:

▶ **Lemma 19.** *For each structure $\mathsf{S} \in \mathsf{F}^*(\mathbf{S})$ there exists a $\mathbb{F}_{\mathcal{HR}}$-term t using only constants from $\mathcal{C} \cup \overline{\mathcal{C}}$, such that (i) $\mathsf{forget}^{\overline{\mathcal{C}}}\left(t^{\mathcal{HR}}\right) = \mathsf{S}$ and (ii) for each element $u \in \mathsf{U}_{\mathsf{S}}$ such that $\gamma = \mathsf{col}_{\mathsf{S}}(u) \in \Gamma^{red} \uplus \Gamma^{green}$ there exists a special constant $\mathsf{c}^{i}_{\gamma} \in \overline{\mathcal{C}}$ such that $\sigma_{\mathsf{S}}(\mathsf{c}^{i}_{\gamma}) = u$.*

Proof *(sketch).* We build t by induction of the derivation of $\mathsf{S} = (\mathsf{U}, \sigma) \in \mathsf{F}^*(\mathbf{S})$. For the base case $\mathsf{S} \in \mathbf{S}$, let $t' \in \mathcal{T}$ be a term such that $\mathsf{S} = t'^{\mathcal{HR}}$. By repeating the $\mathsf{label}(t, n, k, \mathsf{c}^{i}_{\gamma})$ operation, we add a special constant c^{i}_{γ} to each leaf n of t, on the appropriate position $1 \leq k \leq \#\mathsf{r}$, where $\mathsf{lab}(n) = \mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$, such that $\sigma(\mathcal{C}_k) = \{u\}$ and $\gamma \stackrel{\text{def}}{=} \mathsf{col}_{\mathsf{S}}(u) \in \Gamma^{red} \uplus \Gamma^{green}$. The choice of $1 \leq i \leq 2$ is nondeterministic. The result of applying these labeling operations to t' is t. Then, $\mathsf{forget}^{\overline{\mathcal{C}}}\left(t^{\mathcal{HR}}\right) = \mathsf{S}$ and (ii) holds, by construction. For the inductive step, let $\mathsf{S} = (\mathsf{S}_1 \uplus \mathsf{S}_2)_{/\approx}$, where $\mathsf{S}_1, \mathsf{S}_2 \in \mathsf{F}^*(\mathbf{S})$ and $\approx$ is an equivalence relation that is generated by the set of pairs $\{(u_{1i}, u_{2i})\}_{i \in I}$, where I is either $\{1\}$ or $\{1, 2\}$. Let $\gamma_{ji} \stackrel{\text{def}}{=} \mathsf{col}_{\mathsf{S}_j}(u_{ji})$ for all $1 \leq j \leq 2$ and $i \in I$. By the inductive hypothesis, there exist terms t_j and integers $1 \leq k_{ji} \leq 2$, such that $(\mathsf{U}_j, \sigma_j) \stackrel{\text{def}}{=} t_j^{\mathcal{HR}}$ and $\mathsf{forget}^{\overline{\mathcal{C}}}\left(t_j^{\mathcal{HR}}\right) = \mathsf{S}_j$, for all $1 \leq j \leq 2$ and $i \in I$.

1. $I = \{1\}$, i.e., $\approx$ is 1-generated.
   a. $\gamma_{11}, \gamma_{21} \in \Gamma^{red} \uplus \Gamma^{green}$: by the inductive hypothesis (ii), there exist $\mathsf{c}^{i_1}_{\gamma_{11}}, \mathsf{c}^{i_2}_{\gamma_{21}} \in \overline{\mathcal{C}}$ such that $u_{j1} = \sigma_j(\mathsf{c}^{i_j}_{\gamma_{j1}})$, for both $1 \leq j \leq 2$. Suppose that $\mathsf{c}^{\ell}_{\gamma_{11} \uplus \gamma_{21}}$ is visible in t_1 (visibility in t_2 is a symmetric case), for some $1 \leq \ell \leq 2$. Then $\gamma_{11} \uplus \gamma_{21}$ must belong to $\Gamma^{red} \uplus \Gamma^{green}$, by the inductive hypothesis (ii). If $\mathsf{c}^{3-\ell}_{\gamma_{11} \uplus \gamma_{21}}$ is not visible in t_2, suppose first that $\mathsf{c}^{3-\ell}_{\gamma_{11} \uplus \gamma_{21}}$ is visible in t_1. Then, $\mathsf{c}^{i_1}_{\gamma_{11}}$, $\mathsf{c}^{\ell}_{\gamma_{11} \uplus \gamma_{21}}$ and $\mathsf{c}^{3-\ell}_{\gamma_{11} \uplus \gamma_{21}}$ are visible in t_1. By Lemma 15, $\gamma_{11}, \gamma_{11} \uplus \gamma_{21} \in \Gamma^{green}$ and $\gamma_{11} \uplus \gamma_{21}$ occurs 3 times in S, thus contradicting the definition of Γ^{blue}. Hence $\mathsf{c}^{3-\ell}_{\gamma_1 \uplus \gamma_2}$ is not visible in either t_1 or t_2 and $t \stackrel{\text{def}}{=} \mathsf{join}(t_1, t_2, \mathsf{c}^{i_1}_{\gamma_{11}}, \mathsf{c}^{i_2}_{\gamma_{21}})$ is well-defined. Else, if $\mathsf{c}^{3-\ell}_{\gamma_{11} \uplus \gamma_{21}}$ is visible in t_2, then $\gamma_{11} \uplus \gamma_{21}$ occurs 3 times in S, thus contradicting the definition of Γ^{blue}.
   b. $\gamma_{11} \in \Gamma^{blue}$ and $\gamma_{21} \in \Gamma^{red}$ ($\gamma_{11} \in \Gamma^{red}$ and $\gamma_{21} \in \Gamma^{blue}$ is a symmetric case): Let n be the leaf of t_1 such that $\mathsf{lab}(n) = \mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$ and $\mathcal{C}_k$ be a set of constants that are (all) interpreted as u_{11}, for some $1 \leq k \leq \#\mathsf{r}$. Let $\mathsf{c}^{\ell}_{\gamma_{21}}$ be the special constant such that $u_{21} = \sigma_2(\mathsf{c}^{\ell}_{\gamma_{21}})$, for some $1 \leq \ell \leq 2$. We can assume w.l.o.g. that $\mathsf{c}^{\ell}_{\gamma_{21}}$ is not visible in n. If this were not to be the case, then n must have been involved in a previous join of a term t_3 with another term t'_1, such that t_1 is the outcome of this join. In this case, we change the construction, by first joining t_2 with t_3, as in the previous case, then joining the result with t'_1,. Note that this is possible due of the associativity of the $\oplus$ operation. Finally, we define $t \stackrel{\text{def}}{=} \mathsf{append}(t_1, t_2, n, k, \mathsf{c}^{\ell}_{\gamma_{21}})$.
2. $I = \{1, 2\}$, i.e., $\approx$ is 2-generated. By Lemma 18, either one of the following holds:
   a. S_1 and S_2 are of type G: let $\mathsf{c}^{k_{ji}}_{\gamma_{ji}}$ be special constants such that $u_{ji} = \sigma_j(\mathsf{c}^{k_{ji}}_{\gamma_{ji}})$, for all $1 \leq i, j \leq 2$. Then, we define $t \stackrel{\text{def}}{=} \mathsf{join}(t_1, t_2, (\mathsf{c}^{k_i}_{\gamma_{1i}})_{i \in I}, (\mathsf{c}^{k_i}_{\gamma_{2i}})_{i \in I})$ and check that the operation is well-defined, following a similar argument as in case (1a).

b. S_1 and S_2 are of type R: we can assume w.l.o.g. that u_{ii} is the interpretation of a special constant $\mathsf{c}^{j_i}_{\gamma_{ii}}$, for some $1 \leq j_i \leq 2$, where $\gamma_{ii} \in \Gamma^{red}$, for both $i = 1, 2$. Let n_1 be the leaf of t_1 such that $\mathsf{lab}(n_1) = \mathsf{r}(\mathcal{C}_1, \ldots, \mathcal{C}_{\#\mathsf{r}})$ and u_{11} is the interpretation of (all) constants from $\mathcal{C}_{k_1}$, for some $1 \leq k_1 \leq \#\mathsf{r}$. Analogously, we consider n_2 to be the leaf of t_2 and k_2 the position of the constants from its label, that are interpreted as u_{22}. Under similar assumptions as in the case (2a), ensuring that the result is well defined, we let $t \stackrel{\text{def}}{=} \mathsf{forget}^{\mathsf{c}^{j_1}_{\gamma_{11}}}(\mathsf{append}(t_1, \mathsf{label}(t_2, n_2, k_2, \mathsf{c}^{j_1}_{\gamma_{11}}), n_1, k_1, \mathsf{c}^{j_2}_{\gamma_{22}}))$. The only difference with the previous case is that the indices j_1 and j_2 must be different to avoid name clashes, hence we require 2 special constants c^1_γ and c^2_γ, for each color $\gamma \in \Gamma^{red}$. ◀

3.4 Tree-width Bounded Fusion-closed Sets are Context-free

This subsection completes the proof of the first point of Theorem 9. The final ingredient is the following lemma, whose proof relies on the lifting of the construction that simulates the 1- or 2-generated fusion of structures from terms to tree automata recognising sets of terms:

▶ **Lemma 20.** *Let* $\mathbf{S}$ *be a context-free set of structures conforming to some RGB color scheme. Then, the set* $\mathsf{F}^*(\mathbf{S})$ *is context-free.*

Proof of Theorem 9 (1). "⇒" Since $\mathbf{S}$ is a context-free set, it has bounded tree-width. If $\mathsf{F}^*(\mathbf{S})$ has bounded tree-width, then $\mathbf{S}$ conforms to an RGB color scheme, by the combined results of Lemmas 11 and 16. Because $\mathbf{S}$ is context-free, we obtain that $\mathsf{F}^*(\mathbf{S})$ is context-free, by Lemma 20. "⇐" Because each context-free set has bounded tree-width. ◀

3.5 Color Abstractions

This section is concerned with the proof of the second point of Theorem 9. Let $\mathbf{S}$ be a context-free set of structures given by a tree automaton $\mathcal{A}$ over a finite signature $\mathcal{F} \subseteq \mathbb{F}_{\mathcal{HR}}$. In the rest of this section, $\mathbf{S}$, $\mathcal{F}$ and $\mathcal{A}$ are considered to be fixed.

Note that the equivalence between (A) and (B) follows from (A) ⇒ (B) (Lemma 11), (B) ⇒ (C) (Lemma 16) and (C) ⇒ (A) (Lemma 17). Hence, it is sufficient to establish the decidability of the condition (B) for $\mathbf{S}$. To this end, we compute the k-multiset abstraction $(\mathsf{F}^*(\mathbf{S}))^{\sharp k}$, for an arbitrary given integer $k \geq 1$ (note that checking (B) requires $k = 3$). First, we reduce the computation of $(\mathsf{F}^*(\mathbf{S}))^{\sharp k}$ to that of $\mathbf{S}^{\sharp k}$. Second, we sketch the argument behind the effective computability of $\mathbf{S}^{\sharp k}$.

The following lemma shows that, because we are interested only in k-multisets color abstractions, we can restrict fusion to 1-generated equivalence relations, while preserving the k-multiset color abstraction. We denote by $\mathsf{F}^*_1(\mathbf{S})$ the set of structures obtained by taking the closure of $\mathbf{S}$ only with respect to fusions induced by 1-generated matchings.

▶ **Lemma 21.** $(\mathsf{F}^*(\mathbf{S}))^{\sharp k} = (\mathsf{F}^*_1(\mathbf{S}))^{\sharp k}$ *for any set* $\mathbf{S}$ *of structures and integer* $k \geq 1$.

The set $(\mathsf{F}^*_1(\mathbf{S}))^{\sharp k}$ can be computed by a least fixpoint iteration of the following abstract operation on the domain of k-multiset color abstractions. As the later domain is finite, this fixpoint computation is guaranteed to terminate.

▶ **Definition 22.** *The* single-pair multiset fusion *is defined below, for all* $M_1, M_2 \in \mathrm{mpow}(\Gamma)$*:*

$$\mathsf{f}^\sharp_1(M_1, M_2) \stackrel{\text{def}}{=} \{M \in \mathrm{mpow}(\Gamma) \mid \exists \gamma_1 \in M_1 \,.\, \exists \gamma_2 \in M_2 \,.\, \gamma_1 \cap \gamma_2 = \emptyset, \; M = \{\!\{\gamma_1 \cup \gamma_2\}\!\} \cup \textstyle\bigcup_{i=1,2}(M_i \setminus \{\!\{\gamma_i\}\!\})\}$$

Given an integer $k \geq 1$, the single-pair k-multiset fusion *is defined for M_1, $M_2 \in \mathrm{mpow}(\Gamma)$, such that* $\mathrm{card}(M_1) \leq k$ *and* $\mathrm{card}(M_2) \leq k$*:*

$$\mathsf{f}_1^{\sharp k}(M_1, M_2) \stackrel{\text{def}}{=} \{M \mid \exists M' \in \mathsf{f}_1^{\sharp}(M_1, M_2).\ M \subseteq M',\ \mathrm{card}(M) \leq k\}$$

For a set $\mathcal{M}$ of multisets (resp. k-multisets) of colors, let $\mathsf{f}_1^{\sharp *}(\mathcal{M})$ (resp. $\mathsf{f}_1^{\sharp k *}(\mathcal{M})$) be the closure of $\mathcal{M}$ under taking single-pair fusion on multisets (resp. k-multisets).

This operation is used to compute $(\mathsf{F}_1^*(\mathbf{S}))^{\sharp k}$ by a iterating $\mathsf{f}_1^{\sharp k *}$ starting with $\mathbf{S}^{\sharp k}$ until a fixed point is reached. Since there are finitely many colors, the domain of multisets of colors having multiplicity at most k is finite, hence this iteration is guaranteed to compute $\mathsf{f}_1^{\sharp k *}(\mathbf{S}^{\sharp k})$ in finitely many steps.

▶ **Lemma 23.** *$(\mathsf{F}_1^*(\mathbf{S}))^{\sharp k} = \mathsf{f}_1^{\sharp k *}(\mathbf{S}^{\sharp k})$, for any set $\mathbf{S}$ of structures and integer $k \geq 1$.*

The last step concerns the effective computation of $\mathbf{S}^{\sharp k}$, i.e., the set of color multisets of multiplicity at most k that occur in the multiset abstraction of some structure $\mathsf{S} \in \mathbf{S}$. We leverage from the fact that $\mathbf{S}$ is a context-free set described by the set of terms that forms the language of a tree automaton $\mathcal{A}$.

We assume basic acquaintance with *context-free grammars*, i.e., finite sets of rules of the form either $q \leftarrow t[q_1, \ldots, q_k]$ or $\leftarrow q$, where $q, q_1, \ldots, q_k$ denote nonterminals and t is a $\mathbb{F}_{\mathcal{HR}}$-term with variables $q_1, \ldots, q_k$. The language $\mathcal{L}(\Gamma)$ of a grammar Γ is the set of interpretations in $\mathcal{HR}$ of the terms produced by derivations starting with an axiom $\leftarrow q$. It is well known that a tree automaton can be transformed into a context-free grammar having the same language, by turning each transition $(q_1, \ldots, q_k) \xrightarrow{f} q$ into a rule $q \leftarrow f[q_1, \ldots, q_k]$, for $k \geq 1$, and adding an axiom $\leftarrow q$ for each final state q of the tree automaton.

By *first-order logic* we understand the set of formulæ consisting of equalities between variables, relation atoms of the form $\mathsf{r}(x_1, \ldots, x_{\#\mathsf{r}})$, for some relation symbol r, composed via boolean operations and quantifiers. A first-order logic sentence φ (i.e., a formula without free variables) is interpreted over a structure S by the satisfiability relation $\mathsf{S} \models \varphi$, defined inductively on the structure of φ, as usual.

Over words, it is a well-known result that the non-emptiness of the intersection of a context-free set (e.g., given by a context-free grammar) and a regular set (e.g., given by a regular grammar, DFA, or a MSO formula) is decidable. This result has been generalized by Courcelle to context-free grammars over the HR-algebra of structures and FO-definable sets of structures[5]:

▶ **Theorem 24** (Theorem 3.6 in [7])**.** *For each grammar Γ and first-order sentence φ, one can decide the existence of a structure $\mathsf{S} \in \mathcal{L}(\Gamma)$ such that $\mathsf{S} \models \varphi$.*

In order to compute $\mathbf{S}^{\sharp k}$, we build a first-order sentence φ_M for each multiset of colors $M : \Gamma \to [1..k]$ such that $\mathsf{S} \models \varphi_M$ iff $\mathsf{S}^{\sharp k}$. As there are only finitely many such multi-sets M, we are able to construct $\mathbf{S}^{\sharp k}$ by finitely many calls to the above decision procedure. We now state the details for the construction of φ_M. For each color $\gamma \in \Gamma$, we denote by $\varphi_\gamma(X)$ the formula $\bigwedge_{\mathsf{r} \in \mathfrak{C}} \mathsf{r}(x) \wedge \bigwedge_{\mathsf{r} \notin \mathfrak{C}} \neg \mathsf{r}(x)$. We then obtain φ_M as the conjunction of the formulæ $\exists^{=M(\gamma)} x\ .\ \varphi_\gamma(x)$, if $M(\gamma) < k$, and $\exists^{\geq M(\gamma)} x\ .\ \varphi_\gamma(x)$, if $M(\gamma) = k$, for all colors $\gamma \in \Gamma$. As usual, the quantifier $\exists^{=n} x\ .\ \phi(x)$ (resp. $\exists^{\geq n} x\ .\ \phi(x)$) means "there exists exactly n (resp. at least n) elements x that satisfy $\phi(x)$". It is now easy to verify that $\mathsf{S} \models \varphi_M$ iff $M \in \mathsf{S}^{\sharp k}$. This concludes the proof of Theorem 9 (2).

[5] [7, Theorem 3.6] is actually given for Monadic Second Order Logic, which subsumes first-order logic.

4 Conclusions and Future Work

We have defined a non-aggregative and nondeterministic fusion operation on logical structures, that is controlled by a coloring of structures using unary relations. We study the tree-width of the closure of a context-free set under fusion. We prove that it is decidable whether the closure of a context-free set has bounded tree-width. Moreover, if this is the case, we show that the closure set is context-free as well, described by an effectively constructible grammar.

Future work involves considering more general notions of coloring, e.g., coloring functions defined by MSO-definable transductions. Moreover, we plan to investigate generalizations of Theorem 9 for other notions of width, such as generalizations of clique-width and rank-width from graphs and hypergraphs to relational structures.

References

1 Emma Ahrens, Marius Bozga, Radu Iosif, and Joost-Pieter Katoen. Reasoning about distributed reconfigurable systems. *Proc. ACM Program. Lang.*, 6(OOPSLA2):145–174, 2022. doi:10.1145/3563293.

2 Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic and tree update. In Jens Palsberg and Martín Abadi, editors, *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 271–282. ACM, 2005. doi:10.1145/1040305.1040328.

3 Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic as modal logic: completeness and parametric inexpressivity. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 123–134. ACM, 2007. doi:10.1145/1190216.1190236.

4 Luca Cardelli, Philippa Gardner, and Giorgio Ghelli. A Spatial Logic for Querying Graphs. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, July 2002. doi:10.1007/3-540-45465-9_51.

5 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. HAL, 2008. URL: https://inria.hal.science/hal-03367725.

6 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

7 Bruno Courcelle. The monadic second-order logic of graphs VII: graphs as relational structures. *Theor. Comput. Sci.*, 101(1):3–33, 1992. doi:10.1016/0304-3975(92)90148-9.

8 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012. doi:10.1017/CBO9780511977619.

9 Simon Docherty and David J. Pym. Intuitionistic layered graph logic: Semantics and proof theory. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:11)2018.

10 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

11 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

12 Radu Iosif and Florian Zuleger. Expressiveness results for an inductive logic of separated relations. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory (CONCUR 2023)*, volume 279 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2023.20.

13 Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 14–26. ACM, 2001. doi:10.1145/360204.375719.

14 Viktor Kuncak and Martin Rinard. Generalized records and spatial conjunction in role logic. In *Static Analysis*, pages 361–376, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27864-1_26.

15 Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, CSL '01, pages 1–19, 2001. doi:10.1007/3-540-44802-0_1.

16 David J. Pym, Peter W. O'Hearn, and Hongseok Yang. Possible worlds and resources: the semantics of bi. *Theoretical Computer Science*, 315(1):257–305, 2004. Mathematical Foundations of Programming Semantics. doi:10.1016/j.tcs.2003.11.020.

17 Greg Restall. Substructural Logics. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2024 edition, 2024.

18 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029817.

19 D. Seese. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53(2):169–195, 1991. doi:10.1016/0168-0072(91)90054-P.

20 P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993. doi:10.1006/jctb.1993.1027.

Clustering in Varying Metrics

Deeparnab Chakrabarty
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Jonathan Conroy
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Ankita Sarkar
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Abstract

We introduce the *aggregated clustering* problem, where one is given T instances of a center-based clustering task over the same n points, but under different metrics. The goal is to open k centers to minimize an aggregate of the clustering costs – e.g., the average or maximum – where the cost is measured via k-center/median/means objectives. More generally, we minimize a norm Ψ over the T cost values. We show that for $T \geq 3$, the problem is inapproximable to any finite factor in polynomial time. For $T = 2$, we give constant-factor approximations. We also show W[2]-hardness when parameterized by k, but obtain $f(k, T)\text{poly}(n)$-time 3-approximations when parameterized by both k and T.

When the metrics have structure, we obtain efficient parameterized approximation schemes (EPAS). If all T metrics have bounded ε-scatter dimension, we achieve a $(1 + \varepsilon)$-approximation in $f(k, T, \varepsilon)\text{poly}(n)$ time. If the metrics are induced by edge weights on a common graph G of bounded treewidth tw, and Ψ is the sum function, we get an EPAS in $f(T, \varepsilon, \mathsf{tw})\text{poly}(n, k)$ time. Conversely, unless (randomized) ETH is false, any finite factor approximation is impossible if parametrized by only T, even when the treewidth is $\mathsf{tw} = \Omega(\text{poly} \log n)$.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability

Keywords and phrases Clustering, approximation algorithms, LP rounding, parameterized and exact algorithms, dynamic programming, fixed parameter tractability, hardness of approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.19

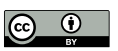
Related Version *Full Version*: `https://arxiv.org/abs/2510.07860` [16]

Funding *Deeparnab Chakrabarty*: Work partially funded by NSF grants CCF-2041920 and CCF-2402571.
Jonathan Conroy: Work partially funded by the U.S. National Science Foundation CAREER Award under the Grant No. CCF-2443017.
Ankita Sarkar: Work partially funded by NSF grants CCF-2041920 and CCF-2402571.

1 Introduction

Clustering problems such as the k-supplier or the k-median problem are classic discrete optimization problems which have formed the bedrock of many problems in operations research. In its simplest form, the objective is to "open" k locations in a metric space such that some function (eg, max or sum) of the distances of clients to the nearest open facility is minimized. Such problems have been extensively studied over the past 50 years [39, 53, 24, 41, 23, 7, 25, 14, 49, 57, 19, 43, 18, 50, 52, 20, 26, 33, 4, 17], and we have a very good understanding on the approximability of many of these problems.

In this paper, we consider clustering problems *when the distance function underlying the metric space can change over time.* To give a toy example, imagine that the clients are households in a New England town, and the distances are travel times over the road network



© Deeparnab Chakrabarty, Jonathan Conroy, and Ankita Sarkar;
licensed under Creative Commons License CC-BY 4.0



in the town. On one hand, travel times may be quite different in January and June given the weather differences over the months. On the other hand, one can't hope to open different locations in different seasons. Our decision ought to take into account the different costs over time, for example by selecting a set of facilities that minimizes the average cost over the year. Although this sounds a natural question, to our knowledge what happens to the complexity of clustering with changing metrics hasn't been studied so far.[1] Formally, we study the following problem.

▶ **Definition 1** (Aggregate Clustering Problems). *We are given $V = F \cup C$, a finite set of n points where C is the set of clients and F is a set of facilities*[2]*. We are given T distance functions $d_t(\cdot,\cdot)$ over $V \times V$ which are symmetric and satisfy triangle-inequality. We are given a parameter $k \in \mathbb{Z}$. Finally, we are given an* aggregator *function $\Psi : \mathbb{R}^T_{\geq 0} \to \mathbb{R}_{\geq 0}$. The goal is to "open" a subset $S \subseteq F$ of k facilities that minimize a certain objective function (depending on the problem variant). The objective functions are:*

- $\Psi\left(\text{cost}_\infty(d_t; S) : t \in [T]\right)$*, where* $\text{cost}_\infty(d_t; S) := \max_{j \in C} d_t(j, S)$*, in the Ψ-aggregate k-supplier problem.*[3]
- $\Psi\left(\text{cost}_1(d_t; S) : t \in [T]\right)$*, where* $\text{cost}_1(d_t; S) := \sum_{j \in C} d_t(j, S)$*, in the Ψ-aggregate k-median problem.*
- $\Psi\left(\text{cost}_z(d_t; S) : t \in [T]\right)$*, where* $\text{cost}_z(d_t; S) := \left(\sum_{j \in C} d_t(j, S)^z\right)^{1/z}$*, in the Ψ-aggregate (k, z)-clustering problem.*

When it is clear from context, we denote the objective function as $\text{aggcost}(\cdot)$.

The above problem also can be thought of as a stochastic/robust version of clustering under uncertainty where the uncertainty is over the distances. Note that when Ψ is the sum-function, we wish to find a solution which minimizes the average k-supplier/k-median cost of the returned solution S, where the average is over the T "scenarios" (we are assuming the full-information setting); when Ψ is the max-function, we get the robust optimization setting. In the literature [5, 52, 26], stochastic/robust clustering has been studied when the metric is fixed but the uncertainty is over the client sets. We can consider a generalized version of aggregate clustering, which models client demands changing over time.

▶ **Definition 2** (Generalized Aggregate Clustering). *Apart from the t distances, we have t weight functions $w_t : C \to \mathbb{R}$. In the objective functions, we modify* costs *by multiplying $d_t(j, S)$'s with $w_t(j)$. More precisely,* $\text{cost}_z(d_t, S) = \left(\sum_{j \in C} w_t(j) \cdot d_t(j, S)^z\right)^{1/z}$ *and* $\text{cost}_\infty(d_t; S) = \max_{j \in C} w_t(j) d_t(j, S)$*. In particular, if $w_t : C \to \{0, 1\}$, then this models having different client sets at different times.*

All our upper bounds apply for generalized aggregate clustering, and all our lower bounds hold even for the vanilla version of aggregate clustering. In fact, in Observation 27 we present a weak equivalence between the two problems, by showing that any algorithm for vanilla aggregate clustering can solve generalized aggregate clustering with $\{0, 1\}$ weights, i.e. aggregate clustering where the client set is allowed to change over time.

---

[1] One may be tempted to simply perform clustering on a single metric space defined by average distances over all months. However, this does not work: the cost of a client at each timestep is computed by take a minimum over a set of facilities, and this minimum is not preserved when we take an average.

[2] F and C need not be disjoint, and one can imagine the case of $F = C$ as a special but instructive case.

[3] We use the notation $d_t(j, S)$ to mean $\min_{i \in S} d_t(j, i)$.

1.1 Our Findings

General Metrics. In its full generality, unfortunately, the aggregate clustering problem becomes very hard when the number of scenarios T becomes 3 or larger. In Theorem 3 we show that it is NP-hard to distinguish between zero and non-zero solutions when $T \geq 3$. To complement this result, we show (in Theorems 7 and 16) that when $T = 2$, both the (generalized) Ψ-aggregate k-median and k-supplier problems have polynomial time $O(1)$-approximations when Ψ is any norm. In fact, for k-supplier, we match the optimal 3-approximation that is known at $T = 1$ [39, 53]. Both the above results, at a very high level, follow from the fact that bipartite matching (or rather, partition matroid intersection) is tractable, while 3D-matching is NP-hard.

Given the above hardness for $T \geq 3$, we attempt to bypass it in two ways: (i) investigating fixed-parameter tractable approximation algorithms, and (ii) imposing additional structure over the metrics in various scenarios.

Constant-factor FPT approximation algorithms. The hardness for $T = 3$ clearly shows one can't expect $f(T) \cdot \text{poly}(n, k)$-time approximation algorithms. Moreover, we show (in Theorem 5) that it is also hard to obtain $f(k) \cdot \text{poly}(n, T)$-time approximation algorithms: we encode the k-hitting set problem as an instance of aggregate clustering, and our lower bound follows from the $W[2]$ hardness of k-hitting set [32, Theorem 13.12]. On the other hand, if we allow k and T to be both fixed parameters, then (in Theorem 19) we obtain an $f(k, T) \cdot \text{poly}(n)$-time $(3 + \varepsilon)$-approximation. The high level idea for this is as follows: Suppose we have T partitions of a universe U, and we must decide if there is a k-sized subset that is a *simultaneous* hitting set for all T partitions. In $(k + 1)^{kT} \cdot \text{poly}(n)$-time, we can guess for each $t \in [T]$ and $i \in [k]$ the part in the t^{th} partition that the i^{th} hitting element hits. Such an idea is the core of many FPT approximation algorithms for clustering [1, 12, 38].

Approximation schemes in structured metrics. To impose structure on the metric, we note that any finite metric d_t is a shortest path metric on an undirected *base graph* $G = (V, E)$ with weights $w_t(e)$ on the edges. Note that our restriction of having the *same* graph G across scenarios is without loss of generality if we allow a complete graph. Furthermore, in many applications, such as the road-network application mentioned in the early paragraphs of the introduction, this indeed is the case where the "weights" change over the seasons but the underlying network remains the same. Can one obtain better algorithmic results exploiting the structure of the underlying undirected base graph G?

Unfortunately, our hardness results apply even for very simple base graphs G. In particular, our reduction from hitting set (Theorem 5, which rules out any finite factor approximation algorithms in $f(k) \cdot \text{poly}(n, T)$ time) can be carried out even when G is a star. Moreover, our reduction from 3D matching (Theorem 3, which rules out any $f(T) \cdot \text{poly}(n, k)$-time finite factor approximations) can be carried out even when G is a grid graph (see Theorem 4).

Nevertheless, we bypass this hardness by designing $1 + \varepsilon$ approximation algorithms in time $f(\varepsilon, k, T) \cdot \text{poly}(n)$ (ie, EPASes in parameters k and T) for a broad class of structured input metrics, which includes (for example) the case when G is a planar graph. More precisely, we design approximation algorithms whenever input metric d_t has *bounded scatter dimension*. This concept was recently introduced by Abbasi *et al.* [1], along with an algorithm demonstrating that metrics of bounded scatter dimension admit $f(\varepsilon, k) \cdot \text{poly}(n)$ time $1 + \varepsilon$-approximation algorithms for many clustering problems. We generalize their ideas to obtain an EPAS for the (generalized) aggregate (k, z)-clustering problem. Bounded scatter dimension captures a rich class of metrics, including doubling metrics, shortest-path metrics on planar

or minor-free graphs [13], and metrics of *bounded highway dimension*[4] – this last class, which was introduced to model transportation networks [3, 2], may be of particular interest in light of our road-network application from earlier. We emphasize that our lower bounds on the star and grid apply to metrics of bounded scatter dimension, so our $\mathrm{FPT}(k, T)$ approximation is essentially the best one could hope to achieve (up to the $1+\varepsilon$ approximation) in this setting.

Additionally, we consider the case when G is a tree, or more generally a bounded-treewidth graph. As bounded-treewidth graphs have bounded scatter dimension, our $f(\varepsilon, k, T) \cdot \mathrm{poly}(n)$-time algorithm from above applies. However, we can do even better – we obtain a $(1+\varepsilon)$-approximation for the sum-aggregate (k, z)-clustering problem in time $f(T, \varepsilon) \cdot \mathrm{poly}(n, k)$, by generalizing a folklore dynamic programming algorithm for k-median on bounded-treewidth graphs [6, 42, 29]. Parameterizing by T is necessary, because of our hardness result on the star. On the other hand, we can avoid parameterizing by k because our hardness for the $T = 3$ case only applies G is a grid graph, which has high treewidth – recall that graphs of bounded treewidth are precisely those graphs which exclude a grid as a minor [54, 28]. In fact, we show in Theorem 4 that our $T = 3$ hardness rules out $f(T) \cdot \mathrm{poly}(n)$-time algorithms for *any* graph G that has sufficiently large treewidth (specifically, $\Omega(\log^{295} n)$), whereas our upper bound in Theorem 24 achieves $f(T) \cdot \mathrm{poly}(n)$ runtime so long as G has treewidth $o(\log n / \log\log n)$ (see Remark 25). Thus, our upper and lower bounds show that the treewidth of the base graph G almost entirely captures the tractability of $\mathrm{FPT}(T)$ algorithms for the sum-aggregate[5] clustering problem (up to some $1+\varepsilon$ approximation factor, and up to some $\mathrm{poly}(\log n)$ gap in the treewidth between our upper and lower bounds).

Open Directions

We end this section mentioning some open directions for future study. If one observes the metrics we use to prove our hardness results, we notice that they "change a lot" as one moves from d_t to d_{t+1}. Can one obtain positive algorithmic results if this change is bounded? Concretely, assume d_t is the shortest path metric on $G_t = (V, E_t)$ and d_{t+1} is the same on $G_{t+1} = (V, E_{t+1})$. What if we forced $|E_t \Delta E_{t+1}| \leq O(1)$? Again taking the road-network example, these could lead to certain road-closures or road-building, but in a single period perhaps not too many edges are added or deleted. We do not know any lower bound for this version, nor do we know how to design algorithms. Structurally, this seems to require an understanding of how shortest paths change with insertion/deletion of edges which may be of independent interest.

Another interesting, but slightly less concrete, direction is to explore *online* algorithms where an algorithm is allowed to change the centers but the distance metric is revealed only after the algorithms decision is made. Can one design "low-regret" algorithms as in [59]? The sum-aggregate objective is in fact the benchmark with which regret is measured, and our results show that approximating this benchmark for general metrics is hard. With this in view, what would be a good definition of regret?

Coming back to aggregate clustering, a concrete open problem is the following. The aforementioned hardness when G is a star actually has $F \neq C$, but the case of $V = C = F$ is also interesting. Are there $f(T) \cdot \mathrm{poly}(n, k)$ approximation or even exact algorithms for Ψ-aggregated k-supplier/median problem when G is a tree? We believe this is an interesting generalization of k-median problem on trees and worthy of exploration.

---

[4] We are not aware of a published proof that highway dimension has bounded scatter dimension, but the claim can be found at timestamp 0:52 in the recorded FOCS 2023 talk on [1] (see link `https://focs.computer.org/2023/schedule/`) and also in [12].

[5] We expect that a DP could be designed for other Ψ-aggregator norms, but in this extended abstract we only consider $\Psi = \mathrm{sum}$.

1.2 Related Works

As noted earlier, approximation algorithms for clustering problems have a rich history and we don't attempt to summarize this. Rather we mention the works most related to the "changing metric" viewpoint.

Deng, Li, and Rabani [33] consider the k-clustering problems in a dynamic setting in which both clients and facilities may change over time (though the metric remains the same). The algorithm is allowed to change the open facilities over time, and the objective is to minimize the clustering costs and the distances that open facilities move over time. Their main results, at a high level, are in lockstep with our first set of results for aggregate clustering (on the tractability of $T = 2$ vs $T \geq 3$ scenarios); though our findings and theirs do not imply one another. Their setting has an ambient metric space (X, d) and, over T timesteps, there are client and facility sets $C_t, F_t \subseteq X$ for each timestep t. The goal is to pick k-sized solutions $S_t \subseteq F_t$ at each time step minimizing a certain objective function.

As mentioned earlier, our problem is related to stachastic/robust clustering problems which consider scenarios where client sets change but the algorithm needs to open the same set of k centers to serve these. This question was studied by Anthony, Goyal, Gupta, and Nagarajan [5] who obtain a $O(\log T + \log n)$-approximation for the robust k-median problem discussed earlier. They also show that stochastic k-center (i.e. stochastic k-supplier with $F = C$) is as hard to approximate as the densest-k-subgraph problem. Makarychev and Vakilian [52] obtain an $\left(e^{O(z)} \frac{\log T}{\log \log T}\right)$-approximation to robust (k, z)-clustering. Along with Chlamtáč, the same authors [26] subsequently generalize this approximation algorithm to clustering with cascaded norm objectives, i.e. objectives of the form $\left(\sum_{t=1}^{T} \left(\sum_{j \in C_t} d(j, S)^z\right)^{q/z}\right)^{1/q}$. When $z \geq q$, they obtain a near-optimal $O\left(k^{O_{z,q}(1)}\right)$-approximation. When $z \leq q < \infty$, they obtain an $O_{z,q}(1)$-approximation.

Another related line of work is *dynamic* algorithms for clustering where clients come and go, and the algorithm needs to update the solution fast and maintain $O(1)$-approximations. There has been a lot of work [47, 22, 31, 36, 8, 10, 46, 37, 11, 9] recently on these probelms. We point the readers to [11] (and references within) for dynamic k-center, and [9] (and references within) for dynamic k-median. It is curious to see if techniques from these can address the questions mentioned in the open-directions paragraph earlier.

2 Hardness of Approximation

▶ **Theorem 3** (Hardness of approximation when $T \geq 3$). *For any homogeneous aggregator Ψ and any $z \in \mathbb{N} \cup \{\infty\}$, there is no $f(T)\text{poly}(n, k)$-time finite-factor approximation for Ψ-aggregate (k, z)-clustering on finite metrics on n-vertices unless $P = NP$. The result holds even for $T = 3$.*

Proof. We show that it is NP-hard to decide whether the optimum value is finite or infinite for any Ψ-aggregate (k, z)-clustering problem. We reduce from the (perfect) 3D Matching problem where, recall, we are given a 3-partite hypergraph $H = (V_1 \uplus V_2 \uplus V_3, E)$ where $|V_1| = |V_2| = |V_3|$ any every hyperedge $e \in E$ intersects every V_t exactly once. The goal is to decide whether or not there exists a *matching* (pairwise disjoint subset) $M \subseteq E$ of hyperedges which spans all vertices. Equivalently, $|M \cap e| = 1$ for all $e \in E$. Given such a 3D Matching instance $(V_1 \uplus V_2 \uplus V_3, E)$, we construct an aggregate clustering instance on $T = 3$ metrics, where the underlying graph is a complete graph G with vertex set E. The client set $C = F = E$ and $k := |V_1| = |V_2| = |V_3|$. For each metric $t \in \{1, 2, 3\}$ and each two distinct clients e, e', we set the distance $d_t(e, e') := 0$ if $e \cap e' \cap V_t \neq \emptyset$, i.e. if they agree on

their vertex in V_t. Otherwise set $d_t(e, e') \coloneqq \infty$. It is easy to see that each d_t is in fact a metric since if e_1, e_2 share the same V_t endpoint as do e_2, e_3, then they must be the same endpoint and thus e_1, e_3 share that as well. It remains to show that the clustering instance admits 0-cost solutions iff the original instance admits a perfect 3D matching.

- Suppose this clustering instance admits a 0-cost solution. Call this solution S. Fix a $t \in \{1, 2, 3\}$ and $v \in V_t$. Let $E(v)$ be the set of hyperedges that contain v. We know by construction that, $\forall e \in E(v), d_t(e, S) = 0$, i.e. $\exists e' \in S$ such that e, e' agree on their vertex in V_t. But then this vertex must be v itself. Thus $e' \in E(v) \cap S$, so $|E(v) \cap S| \geq 1$. Since $|V_t| = |S|$, this means that for each $v \in V_t$, $|E(v) \cap S| = 1$. So S is a perfect 3D matching.
- Suppose M is a perfect 3D matching in the original instance. Fix $t \in \{1, 2, 3\}$ and $e \in C$ such that $e \cap V_t = \{v\}$. We know that $M \cap E(v) \neq \emptyset$, so fix $e' \in M \cap E(v)$. Then $e \cap e' \cap V_t = \{v\}$, i.e. $d_t(e, e') = 0$, and so $d_t(e, M) = 0$. So for each $t \in \{1, 2, 3\}$ and each $e \in C$, we have $d_t(e, M) = 0$. Thus $\text{aggcost}(M) = 0$. ◀

Using techniques from graph drawing literature, one can show that our reduction Theorem 3 holds even when the base graph is a grid graph. Furthermore, we can rule out $\text{poly}(n)$-time algorithms for aggregate clustering whenever the base graph G contains even a relatively small $(\tilde{\Omega}(\log^3 n) \times \tilde{\Omega}(\log^3 n))$ grid minor, and consequently has relatively small treewidth; this follows from the fact that, under the randomized exponential time hypothesis, it is hard to obtain a $2^{o(n)}$-time algorithm for 3D matching [45, 44]. We defer the details to the full version [16].

▶ **Theorem 4.** *Let G be any n-vertex graph with treewidth $\mathsf{tw} = \Omega(\log^{295} n)$. (For example, one may simply take G to be a $\sqrt{n} \times \sqrt{n}$ grid graph.) For any homogenous aggregator Ψ and any $z \in \mathbb{N} \cup \{\infty\}$, there is no $f(T)\text{poly}(n, k)$-time finite-factor approximation for Ψ-aggregate (k, z)-clustering problem when the base graph inducing the metrics is G, unless the randomized exponential time hypothesis is false. This results holds even for $T = 3$.*

Next, we give a different reduction to rule out that FPT algorithms parameterized by k, even when the base graph is a star.

▶ **Theorem 5** (Hardness of approximation on stars in FPT time.)**.** *For any homogenous aggregator Ψ and any $z \in \mathbb{N} \cup \{\infty\}$, there is no $f(k)\text{poly}(n, T)$-time finite-factor approximation algorithms for the Ψ-aggregate (k, z)-clustering problem even when the base graph G inducing the metrics is a star, unless W[2] = FPT.*

Proof. We reduce from k-hitting set, which is W[2] hard when parameterized by k [32]. Suppose we are given an instance of k-hitting set defined by a universe $U = \{v_1, \ldots, v_n\}$ of n points, a set $\mathcal{X} = \{X_1, \ldots, X_m\}$ of m subsets of U, and an integer k; our task is to determine whether there are k points in U that hit all sets of $\mathcal{X}$. We construct an instance of aggregate clustering as follows. Let G be a star graph with a root vertex r connected to n leaves $v_1, \ldots, v_n$. Let the facility set be $F \coloneqq \{v_1, \ldots, v_n\}$, and let the client set (in every metric) be $C = \{r\}$. We define $T \coloneqq m$ distances functions, one for each set $X_t \in \mathcal{X}$, as follows: in the t-th metric, we set the weight of edge (r, v_i) to be 0 if $v_i \in X_t$ and ∞ otherwise.

It remains to show that the Ψ-aggregate (k, z)-clustering problem has finite cost iff there is a hitting set of size k.

- First we show that, if there is a hitting set S of size k, then the $\text{aggcost}(S) = 0$. Indeed, for every $t \in [T]$, there exists some $v_i \in S$ with $v_i \in X_t$ and so $d_t(r, S) \leq d_t(r, v_i) = 0$. In particular, the (k, z)-clustering cost paid by the client set $C = \{r\}$ in each metric is 0, and so the Ψ-aggregate cost is also 0 because Ψ is a norm.

- Similarly, if there is a set S of k facilities such that $\text{aggcost}(S) < \infty$, then we claim S is a hitting set. Indeed, consider some $X_t \in \mathcal{X}$. Since S has finite cost, we know that $d_t(r, S) < \infty$ and in particular there exists some $v_i \in S$ with $d_t(r, v_i) = 0$. By construction, v_i hits X_t. ◀

▶ Remark 6. The above reduction additionally shows that designing *bicriteria* approximation algorithms for aggregate clustering remains hard, due to the hardness of approximation of hitting set [34] – there exists some $\beta = O(\log n)$ such that it is NP-hard to distinguish between the case that there is a hitting set of size k and the case that there is no hitting set of size βk.

3 Constant-factor approximations for $T = 2$ scenarios

In this section we extend the vanilla $O(1)$-approximations for k-supplier and k-median for the case of $T = 2$ scenarios; this contrasts with the inapproximability for $T \geq 3$ (Theorem 3). For simplicity, we restrict our exposition when the aggregator function Ψ is just the sum, but we later point out in Remark 17 why it holds for any homogeneous aggregator such as a norm. In Remark 18 we explain how our algorithms generalize for (k, z)-clustering.

At a high level, both the extensions from vanilla work because (a very simple) matroid *intersection* is polynomial time tractable, while even 3D-matching is NP-hard; the latter was the root of the hardness for $T \geq 3$ aggregate clustering. We begin with the k-supplier problem which explains the previous line, and then show how ideas from the "matroid-median" problems solves the aggregate k-median problem for $T = 2$.

3.1 3-approximation for Aggregate k-Supplier

Let $\texttt{OPT}$ be the optimal set of k-suppliers which minimizes $\text{cost}_\infty(d_1; \texttt{OPT}) + \text{cost}_\infty(d_2; \texttt{OPT})$, where recall $\text{cost}_\infty(d_t; \texttt{OPT}) = \max_{j \in C} d_t(j, \texttt{OPT})$. Let us denote $\text{cost}_\infty(d_t; \texttt{OPT})$ as $\texttt{opt}_t$ for $t \in \{1, 2\}$. Note that this can be guessed in poly(n)-time (one could make this poly-logarithmic in n using "binary search" techniques) and so we assume we know $\texttt{opt}_1$ and $\texttt{opt}_2$. We now describe an algorithm, very similar to the one in [39], which returns a subset $\texttt{ALG}$ of facilities with $\text{cost}_\infty(d_t; \texttt{ALG}) \leq 3\texttt{opt}_t$ for $t \in \{1, 2\}$. This implies a 3-approximation for the aggregate k-supplier problem for the $T = 2$ case for any homogeneous aggregator Ψ.

For $t \in \{1, 2\}$, define $B_t(a, r) = \{b \in F \cup C : d_t(a, b) \leq r\}$. We run the Hochbaum-Shmoys [39] filtering algorithm on each metric d_t with parameter $\texttt{opt}_t$. To briefly describe this: (i) initially all clients are "uncovered", (ii) pick an arbitrary *uncovered client* $j \in C$ and add it to a set R_t of representatives, define $B_t(j, 2\texttt{opt}_t) \cap C$ to be j's "children", and mark j and all its children as "covered", and (iii) continue the above till all clients covered. The main observations, which are easy to check using triangle inequality and the fact that $\texttt{opt}_t$'s were correct guesses, are the following for $t \in \{1, 2\}$:

- The collection of balls $\{B_t(j, \texttt{opt}_t) \cap F\}_{j \in R_t}$ are pairwise disjoint.
- For any $j \in R_t$, $B_t(j, \texttt{opt}_t) \cap \texttt{OPT} \neq \emptyset$. The above two imply $|R_t| \leq k$.
- For every $v \in C$, $d_t(v, R_t) \leq 2\texttt{opt}_t$.

We now construct a solution $\texttt{ALG}$ of size $\leq k$ such that for any $t \in \{1, 2\}$ and any $j \in R_t$, we have $\texttt{ALG} \cap B_t(j, \texttt{opt}_t) \neq \emptyset$; that is, $\texttt{ALG}$ hits every ball in the above two collections. This can be solved using *matroid intersection* since there are only two collections of disjoint balls. Once we have such an $\texttt{ALG}$, it is easy to see using triangle inequality that, for any $t \in \{1, 2\}$, $d_t(v, \texttt{ALG}) \leq d_t(v, j) + d_t(j, \texttt{ALG}) \leq 3\texttt{opt}_t$ where j was v's representative in R_t.

To give more details on how to find $\texttt{ALG}$, for $t \in \{1,2\}$ define the partition $\mathcal{P}_t$ of F formed by $\{B_t(j, \texttt{opt}_t) \cap F\}_{j \in R_t} \cup Z_t$, where Z_t are all the facilities of F not in any ball. We define the "budget" of each part to be $|\{B_t(j, \texttt{opt}_t) \cap F\}| - 1$ – this is the number of facilities $\texttt{ALG}$ can "leave out" – and set the budget of Z_t to be $|Z_t|$. We say $S \subseteq F$ is independent in partition matroid t if it picks at most budget from each part; we seek the set $S \subseteq F$ of the largest cardinality which is independent in both partition matroids. We return $\texttt{ALG} := F \setminus S$. By design, $\texttt{ALG}$ hits every $B_t(j, \texttt{opt}_t)$ for $t \in \{1,2\}$ and $j \in R_t$. Since $F \setminus \texttt{OPT}$ is a candidate S which is in the intersection of the two partition matroids, we get $|\texttt{ALG}| = |F| - |S| \leq |F| - |F \setminus \texttt{OPT}| = |\texttt{OPT}| \leq k$.

In sum, we can find a solution $\texttt{ALG}$ such that $\text{cost}_\infty(d_t; \texttt{ALG}) \leq 3\texttt{opt}_t$ for $t \in \{1,2\}$, and thus $\Psi(\text{cost}_\infty(d_1; \texttt{ALG}), \text{cost}_\infty(d_2; \texttt{ALG})) \leq 3\Psi(\texttt{opt}_1, \texttt{opt}_2)$. This completes the proof of the following theorem.

▶ **Theorem 7.** *When $T = 2$, there is a* poly(n)*-time 3-approximation algorithm for Ψ-aggregate k-supplier on n vertices for any homogeneous aggregator Ψ .*

3.2 $O(1)$-approximation for Aggregate k-Median

To obtain an constant approximation for the k-median problem one has to work a bit harder, but the underlying idea behind tractability is still the tractability of matroid intersection. More precisely, it is the integrality of a polytope defined by two laminar set systems. At a high level, such an issue arises when one studies the *matroid median* problem where there is only one metric but one has the extra constraint that the set of facilities opened must be an independent set of a certain matroid. One solves this problem by rounding a solution to an LP-relaxation where matroid intersection (or rather the integrality fact mentioned above) forms a core component with the given matroid being one matroid, and the other formed via "filtering" technique a la [55, 23]. In our case the situation is similar at such a high-level – the two "matroids" are formed by the filtering ideas for the $T = 2$ scenarios – but the details do need working out, and we show this below. We use the framework set by Swamy [57] for the matroid-median problem, but other frameworks (such as the iterated rounding framework of [43]) could possibly lead to better approximation factors; in this work, we didn't optimize the latter.

Linear Programming Relaxation

We begin by writing a linear programming relaxation for our problem. Recall that, in the standard linear programs for k-median, matroid median, etc. [23, 48, 57, 33], variables of the form $x(i, v)$ denote whether or not the client v uses the facility i, so that v's share of the cost is $\sum_{i \in F} d(v, i)x(i, v)$. In an integral solution $S \subseteq F$, each v has a unique $i_v \in S$ such that $x(i_v, v) = 1$, and $d(i_v, v) = d(v, S)$.

In our problem, we need *two* sets of such x-variables, because given an integral solution $S \subseteq F$, a client v can use different facilities under different metrics; that is, there can be distinct $i_1, i_2 \in S$ s.t. $d_1(v, S) = d_1(v, i_1)$ and $d_2(v, S) = d_2(v, i_2)$. So for each $t \in \{1,2\}$, we define variables $\{x_t(i, v)\}_{v \in C, i \in F}$ denoting whether or not the client v uses the facility i under the metric d_t. We also have variables $\{y(i)\}_{i \in F}$ which, like in the standard k-median LP [23, 25], denote whether or not the facility i is picked into our solution. This allows us to write linear constraints similar to the standard LP for k-median. We use lp to refer to the fractional optimum of the linear program relaxation.

$$\text{minimize: } \sum_{v\in C}\sum_{i\in F} d_1(i,v)x_1(i,v) + \sum_{v\in C}\sum_{i\in F} d_2(i,v)x_2(i,v) \tag{LP}$$

$$\sum_{i\in F} y(i) \le k \tag{LP1}$$

$$x_t(i,v) \le y(i) \qquad \forall t\in\{1,2\}, v\in C, i\in F \tag{LP2}$$

$$\sum_{i\in F} x_t(i,v) = 1 \qquad \forall t\in\{1,2\}, v\in C \tag{LP3}$$

$$0 \le x_t(i,v), y(i) \le 1 \qquad \forall t\in\{1,2\}, v\in C, i\in F$$

Given a solution (x_1, x_2, y) of LP, we first make the following assumption which follows from *facility splitting* arguments due to Chudak and Shmoys [27] (see also [58])

▶ **Assumption 8.** $\forall t\in\{1,2\}, v\in C, i\in F,\ x_t(i,v)\in\{0, y(i)\}$.

We now perform a *filtering step* inherent to almost all LP-rounding algorithms for k-median. For our problem, as in the k-supplier problem from previous section, we get *two* sets of "representatives" instead of one that k-median rounding algorithms get.

Filtering. For $t\in\{1,2\}$ and any client $v\in C$, define $C_t(v) := \sum_{i\in F} d_t(i,v)x_t(i,v)$, i.e. the cost paid by v under metric d_t. Our filtering algorithm begins with all clients in an *uncovered* set U. We pick $j\in U$ with the smallest $C_t(j)$, and call it a *representative under* d_t, adding it to the set Reps_t. Every *uncovered* client v that is within distance $4C_t(v)$ of j becomes a *child of* j *under* d_t, forming the set $\mathsf{child}_t(j)$. All of $\mathsf{child}_t(j)$ is then considered covered. We repeat this until all clients are covered. So we get $\mathsf{Reps}_t\subseteq C$ that is *well-separated*, i.e. any two distinct $j,j'\in\mathsf{Reps}_t$ have $d_t(j,j') > 4\max\{C_t(j), C_t(j')\}$. Reps_t also induces a partition $\{\mathsf{child}_t(j)\}_{j\in\mathsf{Reps}_t}$ of C. The following lemma is standard (see the full version [16] for a proof), and thereafter, we focus on Reps_1 and Reps_2, and seek a constant-factor approximation on those clients only.

▶ **Lemma 9.** *Consider $S\subseteq F$ and $\alpha\ge 1$, such that $\forall t\in\{1,2\}$, $\sum_{j\in\mathsf{Reps}_t} |\mathsf{child}_t(j)|\, d_t(j,S) \le \alpha\cdot\sum_{v\in C} C_t(v)$. Then* $\mathrm{aggcost}(S)\le(4+\alpha)\cdot\mathtt{opt}$.

The rounding algorithms for many k-median algorithms first round to a half-integral solution and then to integral. We follow the same route, and in particular, follow Swamy's framework [57]. We begin with the following definitions for each $t\in\{1,2\}, j\in\mathsf{Reps}_t$; as in the previous section, we use $B_t(a,r) = \{b\in F\cup C : d_t(a,b)\le r\}$. An illustrative figure for these appears in Swamy's work [57, Figure 1].

- $F_j^t := \left\{i\in F : d_t(i,j) = \min_{j'\in\mathsf{Reps}_t} d_t(i,j')\right\}$ (breaking ties arbitrarily)
- $B_j^t := B_t(j, 2C_t(j))\cap F$. Notice that, by construction of Reps_t, $B_j^t\subseteq F_j^t$ (also follows from Fact 10).
- $\gamma_j^{(t)} := \min_{i\notin F_j^t} d_t(i,j)$, and $G_j^t := \left\{i\in F_j^t : d_t(i,j)\le\gamma_j^{(t)}\right\}$.

We obtain the following a la [57] (see the full version [16] for a proof).

▶ **Fact 10.** *For every $t\in\{1,2\}, j\in\mathsf{Reps}_t$, we have*

- $y(B_j^t) := \sum_{i\in B_j^t} y(i)\ge\frac{1}{2}$
- $B_j^t\subseteq G_j^t$
- *Suppose $\gamma_j^{(t)} = d_t(i_\gamma, j)$, s.t. $i_\gamma\in F_\ell^t$. Then $\forall i\in B_\ell^t, d_t(i,j)\le 3\gamma_j^{(t)}$.*

Also note that by design the F_j^t's (and therefore the G_j^t's) are pairwise disjoint for a fixed t when we consider $j \in \mathsf{Reps}_t$. These play the role of the "partitions" as in the k-supplier problem, and our tractability follows because we have only two t's. Instead of finding the "largest cardinality independent set", as we did for the k-supplier problem, we instead find a point maximizing a suitable linear function. Towards this, we describe a "linear function" that acts as a proxy for $C_t(j)$'s.

For fractional facility masses $z \in [0,1]^F$, $\forall t \in \{1,2\}$, and $\forall j \in \mathsf{Reps}_t$, let $z(G_j^t) := \sum_{i \in G_j^t} z(i)$. Then we have

$$T_t(z,j) := \sum_{i \in G_j^t} d_t(i,j) z(i) + 3\gamma_j^{(t)} \max\left\{0, 1 - z(G_j^t)\right\}, \text{ and}$$
$$T_t(z) := \sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)|\, T_t(z,j)\,.$$

The above definition is quite similar to Swamy's definition which may be taken as the $T = 1$ case, with one difference that we have the "max with 0". When $T = 1$, one can assert $z(G_j) \le 1$ but with different t's, it may be that $z(G_j^2) > 1$ because in metric 1 we need to open (fractionally) a lot of facilities. Nevertheless, even with the "max" function the following two lemmas similar to those in [57] hold. Their proofs are in the full version [16].

▶ **Lemma 11.** *Consider $z \in [0,1]^F$ s.t. $\forall t \in \{1,2\}, j \in \mathsf{Reps}_t$, $z(B_j^t) \ge \frac{1}{2}$. Then j's assignment cost under d_t is at most $T_t(z,j)$. That is, for every $t \in \{1,2\}, j \in \mathsf{Reps}_t$, we can assign variables $\{0 \le x'_t(i,j) \le z(i)\}_{i \in F}$ so that $\sum_{i \in F} d_t(i,j) x'_t(i,j) \le T_t(z,j)$.*

▶ **Lemma 12.** *For each $t \in \{1,2\}$, and each $j \in \mathsf{Reps}_t$, $T_t(y,j) \le 3 \cdot C_t(j)$.*

Rounding to Half-integral Solution. We are now ready to describe the algorithm which is encapsulated in the following lemma.

▶ **Lemma 13.** *There is a polynomial time algorithm that yields a half-integral solution $(\hat{x}_1, \hat{x}_2, \hat{y})$ of LP s.t. $\forall t \in \{1,2\}$, $\sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)| \sum_{i \in F} d_t(i,j) \hat{x}_t(i,j) \le 6 \cdot \sum_{v \in C} C_t(v)$.*

Our goal now becomes to find a half-integral $\hat{y} \in \left\{0, \frac{1}{2}, 1\right\}^F$ so that, for each $t \in \{1,2\}$, $T_t(\hat{y}) \le 6 \cdot \sum_{v \in C} C_t(v)$. If we are able to do so, then by Lemma 11 we would be done. To find such a half-intergral solution, we define the following polytope and linearization of T_t's. The polytope $\mathcal{P}$ has variables $(z, \lambda_1, \lambda_2) \in \mathbb{R}_{\ge 0}^{F \cup \mathsf{Reps}_1 \cup \mathsf{Reps}_2}$. The auxiliary λ_t variables serve to linearize the max terms in the $T_t(z)$'s, as the constraints in $\mathcal{P}$ ensure that $\lambda_t(j) \ge \max\left\{0, 1 - z(G_j^t)\right\}$.

$$\mathcal{P} := \Bigg\{ (z, \lambda_1, \lambda_2) \in \mathbb{R}_{\ge 0}^{F \cup \mathsf{Reps}_1 \cup \mathsf{Reps}_2} : z(F) \le k;\ \forall t \in \{1,2\}, \forall j \in \mathsf{Reps}_t,$$
$$z(B_j^t) \ge \frac{1}{2},\ z(G_j^t) + \lambda_t(j) \ge 1 \Bigg\}$$

The following follows from the presence of only "two partitions" plus noting the λ's don't bother much since they are identity.

▷ Claim 14. $\mathcal{P}$ has half-integral extreme points.

Proof. Consider the constraint matrix M of $\mathcal{P}$. Since the constant terms in the constraints are half-integral, it suffices [40] to show that M is totally unimodular (TU). Indeed, M looks $\left[\begin{array}{c|c} A & \begin{matrix} I \\ 0 \end{matrix} \end{array}\right]$ where we have the A-matrix corresponding to the z-variables, and the I corresponding to the λ_t-variables. The dimension of I is $|\mathsf{Reps}_1| + |\mathsf{Reps}_2|$. The matrix A is the incidence matrix of two laminar systems – in fact, it is two partitions coarsened by the universe. Such a system is TU [57]. A TU matrix padded with columns with at most one 1 in them remains TU. ◁

Over $\mathcal{P}$, we can then consider the *minimization* of the following linearization of the $T_t(z)$'s: $\forall t \in \{1,2\}, \forall j \in \mathsf{Reps}_t$: $W_t(z,\lambda_t,j) := \sum_{i \in G_j^t} d_t(i,j) z(i) + 3\gamma_j^{(t)} \lambda_t(j)$, and

$$W_t(z,\lambda_t) := \sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)|\, W_t(z,\lambda_t,j)\,.$$

The following claim encapsulates that we can find an extreme point of (any polytope) $\mathcal{P}$ which is a 2-approximation to both W_t's *simultaneously* in a sense made clear below. This allows us to get the desired $\hat{y}$ (as we explain better in the proof of Lemma 13).

▷ Claim 15. Consider a polytope $\mathcal{P} \subseteq \mathbb{R}^m_{\geq 0}$, linear functions $W_1, W_2 : \mathbb{R}^m_{\geq 0} \to \mathbb{R}_{\geq 0}$, and a point $p \in \mathcal{P}$. There is a polynomial time algorithm which returns an extreme point $\hat{p}$ of $\mathcal{P}$ such that $W_t(\hat{p}) \leq 2W_t(p)$ for $t \in \{1,2\}$.

Proof. By Carathéodory's theorem [15, 56], we can write p as a linear combination of at most $(m+1)$ extreme points of $\mathcal{P}$. More precisely, we can find, in polynomial time, a subset $\mathcal{E}$ of extreme points such that $|\mathcal{E}| \leq m+1$ and $p = \sum_{q \in \mathcal{E}} \mu_q \cdot q$ for some μ_q's which form a probability distribution D_p. We claim that one of the $q \in \mathcal{E}$ is the desired point, and we can find which one by enumeration. To see why, observe that for $t \in \{1,2\}$, $\mathbb{E}_{\hat{p} \sim D_p}[W_t(\hat{p})] = W_t(p)$, and so by Markov, $\Pr_{\hat{p} \sim D_p}[W_t(\hat{p}) > 2W_t(p)] < \frac{1}{2}$. So, by union bound $\Pr_{\hat{p} \sim D_p}[\exists t \in \{1,2\} :\ W_t(\hat{p}) > 2W_t(p)] < 1$ implying $\Pr_{\hat{p} \sim D_p}[W_t(\hat{p}) \leq 2W_t(p),\ \ t \in \{1,2\}] > 0$ which proves the above claim. ◁

Proof of Lemma 13. Given our initial solution (x_1, x_2, y) of LP, set variables $\lambda_t(j) := \max\left\{0, 1 - y(G_j^t)\right\}$ for each $t \in \{1,2\}$, $j \in \mathsf{Reps}_t$. By Fact 10, this gives us $(y, \lambda_1, \lambda_2) \in \mathcal{P}$. So by Claim 14 and Claim 15, we can obtain a half-integral $(\hat{y}, \hat{\lambda}_1, \hat{\lambda}_2) \in \mathcal{P}$ such that $W_1(\hat{y}, \hat{\lambda}_1) \leq 2W_1(y, \lambda_1)$, and $W_2(\hat{y}, \hat{\lambda}_2) \leq 2W_2(y, \lambda_2)$. So by Lemma 11, for each $t \in \{1,2\}$ we can construct $\hat{x}_t$'s such that,

$$\sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)| \sum_{i \in F} d_t(i,j) \hat{x}_t(i,j) \leq T_t(\hat{y}) \leq W_t(\hat{y}, \hat{\lambda}_t) \leq 2W_t(y, \lambda_t)$$

where the penultimate step follows from the constraints in $\mathcal{P}$. By our construction of the starting λ_t's, i.e. the constraints enforcing $\lambda_t(j) \geq \max\{0, 1 - z(G_j^t)\}$, $W_t(y, \lambda_t) = T_t(y)$. So we have, by Lemma 12, that the above can be upper bounded by $6 \sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)|\, C_t(j) \leq 6 \sum_{v \in C} C_t(v)$. ◀

Rounding to Integral solution. We round $(\hat{x}_1, \hat{x}_2, \hat{y})$ to an integral solution, losing a constant factor, and this is very similar to the ideas in [23, 57]; here also we follow the latter's framework. We sketch the main ideas here and defer the details to the full version [16].

The first step is another *filtering* step. For $t \in \{1,2\}$ and $j \in \mathsf{Reps}_t$, define $S_j^t := \{i \in F\ :\ \hat{x}_t(i,j) > 0\}$ and let $\hat{C}_t(j) := \sum_{i \in F} \hat{x}_t(i,j) d_t(i,j)$; note that $\sum_{j \in \mathsf{Reps}_t} |\mathsf{child}_t(j)| \hat{C}_t(j) \leq 6 \sum_{v \in C} C_t(v)$. Now we define a subset of "super-representatives": we pick $\ell \in \mathsf{Reps}_t$ with

smallest $\hat{C}_t(\ell)$ and add it to Super_t removing every j with $S^t_j \cap S^t_\ell \neq \emptyset$ from consideration. In the end, we have S^t_j's pairwise disjoint for Super_t's and each $j \in \mathsf{Reps}_t$ shares a facility with a super-representative in Super_t.

Next, as in the rounding to the half-integral case, we define a particular linear function which works as a proxy for $\hat{C}_t(j)$. To define this proxy, for every $j \in \mathsf{Reps}_t$ call the two facilities in S^t_j primary and secondary in ascending order of distance (duplicating facilities if needed). For $t \in \{1,2\}$ and $j \in \mathsf{Reps}_t$, let $\ell \in \mathsf{Super}_t$ be the super-representative which j shares a facility with, and define

$$A_t(z,j) := \begin{cases} \sum_{i \in S^t_\ell} d_t(i,j) z(i) & \text{if } \mathsf{prim}_t(j) \in S^t_\ell \\ \sum_{i \in S^t_\ell} d_t(i,j) z(i) + (d_t(\mathsf{prim}_t(j), j) - d_t(\mathsf{sec}_t(j), j)) \cdot z(\mathsf{prim}_t(j)) & \text{otherwise} \end{cases}$$

It's instructive to think of all $z(i)$'s as $1/2$; in that case $A_t(z,j)$ is either the average of j's distance to ℓ's facilities when j's closest facility is shared with ℓ, or it is the average of j's distance to it closest facility and distance to the other facility that ℓ goes to whom j doesn't share. The importance of the above function is captured in a couple of observations: (A) for any t and any $j \in \mathsf{Reps}_t$, we can upper bound $A_t(\hat{y}, j) \leq 2\hat{C}_t(j)$, that is, $A_t(\hat{y}, j)$ isn't too bad a proxy, and perhaps more usefully (B) given any z such that $z(S^t_\ell) = 1$ for all *super representatives*, we can find an assignment of any $j \in \mathsf{Reps}_t$ to facilities with connection cost $\leq A_t(z,j)$. These two facts, and the fact that we can minimize linear functions over the intersection of two partition matroids (defined by the S^t_ℓ's for $t \in \{1,2\}$ and $\ell \in \mathsf{Super}_t$), gives us an $O(1)$-approximation. This is because the relevant polytope has integral extreme points (see the full version [16] for a proof). Observation (A) holds because we can charge j's journey to ℓ's "other facility" to ℓ's connection cost which is smaller than j's cost by design; this uses half-intergrality. The details are in the full version [16]. Observation (B) is immediate if $\mathsf{prim}_t(j) \in S^t_\ell$ or if $z(\mathsf{prim}_t(j)) = 0$; otherwise, if $z(\mathsf{prim}_t(j)) = 1$ then j will travel to its closest facility while $A_t(z,j)$ will be average of two number which are larger than the minimum. The statement is in fact true for fractional z's as well and the proof is in the full version [16].

All in all, we get the following theorem; the factor 28 could definitely be improved, but perhaps no better than 8 using these methods.

▶ **Theorem 16.** *When $T = 2$, there is a polynomial time 28-approximation algorithm for sum-aggregate k-median.*

▶ Remark 17 (Generalizing to arbitrary norm aggregators). In the above proof, we note that the solution S we return has the property that $\mathrm{cost}(d_t; S) \leq 28 \sum_{j \in C} C_t(j)$, for $t \in \{1,2\}$. This allows us to generalize the above theorem for any aggregator Ψ which is a norm. Instead of a linear program, we would have a *convex* program. More precisely, we have variables C_1, C_2 where $C_t := \sum_{j \in C} C_t(j)$ and the objective would minimize $\Psi(C_1, C_2)$. By the property of our rounding and the homogeneity of Ψ, we would get that $\Psi(\mathrm{cost}(d_1; S), \mathrm{cost}(d_2; S)) \leq 28\Psi(C_1, C_2) \leq 28\mathtt{opt}$.

▶ Remark 18 (Generalizing (k,z)-clustering). The above theorem focused on the k-median problem. However, the same methodology also gives an $O(1)$-approximation for the Ψ-aggregated (k,z)-clustering problem due to the fact that we only use triangle inequalities over "bounded number of hops". This is a folklore observation (see, for instance, Footnote 1 in [21]). To obtain this, first we replace $d_t(i,j)$ with $d_t(i,j)^z$ in the linear/convex program. Next, we use the "relaxed triangle inequality", which follows from Lemma A.1 in [51], that says if we have $r+1$ points $a_1, \ldots, a_{r+1}$, then $d(a_1, a_{r+1})^z \leq r^{z-1} \sum_{i=1}^{r} d(a_i, a_{i+1})^z$. In our proof

above, we never invoke the triangle inequality on more than 4 points, and thus, everything goes through with a "hit" of 3^{z-1}. Note, though, that the definition of the proxy function $T_t(z, j)$ would have 3 replaced by 3^z. In the end, we would get a $O(1)^z$ approximation to the sum of the zth powers, and since we take the zth root, this gives a $O(1)$-approximation.

4 FPT(k, T) constant factor approximation

▶ **Theorem 19.** *For any homogenous aggregator Ψ, any $z \in \mathbb{N} \cup \{\infty\}$, and any constant error parameter $\varepsilon > 0$, there is an algorithm that computes a $(3+\varepsilon)$-approximate solution to Ψ-aggregate (k, z)-clustering in time $O(\varepsilon^{-1}k)^{O(kT)} \cdot 2^{O(k^2T^2)} \cdot \text{poly}(n)$.*

Proof. We adapt an existing FPT 3-approximation [30, Section 2.1] for standard (k, z)-clustering (i.e. $T = 1$). Fix an error parameter $\varepsilon > 0$. In our aggregate cluster instance, let C denote the client set and let F denote the facilities. For each scenario $t \in [T]$ (defined by a metric d_t and a client weights w_t), we first compute a *coreset* $C_t \subseteq C$ of size $O(\varepsilon^{-2}k \log n)$ [35]; that is, obtain a new client set C_t and new weights w'_t on C_t with the property that for any subset of facilities $S \subseteq F$,

$$\sum_{u \in C_t} w'_t(u) \cdot d_t(u, S) \in (1 \pm \varepsilon) \cdot \sum_{v \in C} w_t(v) \cdot d_t(v, S)\,.$$

Henceforth we work with the coresets C_t rather than the entire client set C in each metric.

Let $\texttt{OPT} \subseteq F$ be the optimal solution to the aggregate clustering instance, with (unknown) centers $o_1, \ldots, o_k$. For an $i \in [k]$ and $t \in [T]$, let $C_{i,t}$ be the cluster served by o_i under d_t. Define the client $\ell_{i,t}$ to be the closest client (in the coreset C_t) to o_i, that is $d_t(o_i, C_{i,t}) = d_t(o_i, \ell_{i,t})$, and define $r_{i,t} \coloneqq d_t(o_i, \ell_{i,t})$.

For each scenario $t \in [T]$ and index $i \in [k]$, we can guess (by enumeration) the client $\ell_{i,t}$ in time $\binom{O(\varepsilon^{-2}k \log n)}{k}$. We can't guess $r_{i,t}$ exactly, but we can guess it up to the closest power of $(1+\varepsilon)$: we assume WLOG (up to a $1+\varepsilon$ loss in the approximation factor) that the aspect ratio of d_t is $\text{poly}(n)$, and so bucketing pairwise distances of d_t by powers of $(1+\varepsilon)$ reduces the number of possible distances to $O(\log_{1+\varepsilon} n) = O(\varepsilon^{-1} \log n)$. Overall, in time $\left(\binom{O(\varepsilon^{-2}k \log n)}{k} \cdot O(\varepsilon^{-1} \log n)^k\right)^T$, we can guess each $\ell_{i,t}$ and $r_{i,t}$. By Fact 26, this overhead is $O(\varepsilon^{-1}k)^{O(kT)} \cdot 2^{O(k^2T^2)} \cdot \text{poly}(n) = O_{\varepsilon,k,T}(1) \cdot \text{poly}(n)$.

Suppose our guesses are correct. For each $i \in [k]$, consider the set F_i containing the facilities that are within distance $r_{t,t}$ of $\ell_{i,t}$ in every metric d_t, i.e. $F_i \coloneqq \cap_{t=1}^{T} B_t(\ell_{i,t}, r_{i,t}) \cap F$. If our guesses are correct, then F_i is non-empty since o_i lies in this intersection. Let our solution $\texttt{ALG}$ consist of an arbitrary $f_i \in F_i$ for each $i \in [k]$. For a client v in the cluster $C_{i,t}$, we have $d_t(o_i, \ell_{i,t}) \le d_t(v, o_i)$ by definition of $\ell_{i,t}$; and also $d_t(\ell_{i,t}, f_i) \le (1 + O(\varepsilon)) \cdot r_{i,t} = (1 + O(\varepsilon)) \cdot d_t(\ell_{i,t}, o_i)$. Using these, we get

$$\begin{aligned} d_t(v, \texttt{ALG}) &\le d_t(v, f_i) \le d_t(v, o_i) + d_t(o_i, \ell_{i,t}) + d_t(\ell_{i,t}, f_i) \\ &\le d_t(v, o_i) + (2 + O(\varepsilon)) \cdot d_t(o_i, \ell_{i,t}) \le (3 + O(\varepsilon)) \cdot d_t(v, o_i) \end{aligned}$$

for every $v \in C$ and $t \in [T]$. Since the (k, z)-clustering cost and the aggregator Ψ are both homogeneous functions, this yields a $(3 + O(\varepsilon))$-approximation. Rescaling ε by a constant yields the desired result. ◀

In the special case of k-supplier, we have a simpler $(3 + \varepsilon)$-algorithm that doesn't utilize coresets, and runs in $O(\varepsilon^{-O(T^2)}) \cdot O(k)^{O(kT)} \cdot \text{poly}(n)$ time. We present it in the full version [16].

5 FPT Approximations in Well-structured Metrics

5.1 Metrics with Bounded Scatter Dimension

In this section, we study aggregate clustering in the case where all the T metrics have bounded *ε-scatter dimension*. This notion of metric dimension was originally introduced by [1] to obtain $O_{k,\varepsilon}(1) \cdot \text{poly}(n)$-time $(1+\varepsilon)$-approximation algorithms (ie, efficient parameterized approximation schemes, or EPASes for short) for a large class of center-based k-clustering problems, for example the $T = 1$ case of (k, z)-clustering.

▶ **Definition 20** (ε-scattering, ε-scatter dimension [1, Definition IV.1]). *An ε-scattering in a metric space $M = (C \cup F, d)$ is a sequence $(x_1, p_1), (x_2, p_2), \ldots, (x_l, p_l)$ s.t.*

- *$\forall i \in [l]$, $x_i \in F$ and $p_i \in C$;*
- *[covering] $\forall 1 \le j < i \le l$, $d(x_i, p_j) \le 1$; and*
- *[ε-refutation] $\forall i \in [l]$, $d(x_i, p_i) > 1 + \varepsilon$.*

The ε-scatter dimension of a family of metric spaces $\mathcal{M}$ is the maximum length of any ε-scatter of an $M \in \mathcal{M}$.

A key component of the ($T = 1$) EPASes of [1] is an EPAS for the special case of unweighted k-supplier. We recall their algorithm, following their presentation. At a high level, they maintain a k-sized solution $X \subseteq F$, and while this solution is not a $(1+\varepsilon)$-approximation, clients with high cost are identified to guide a recomputation of X; the bounded ε-scatter dimension is used to show a bound on the number of recomputations needed. In more detail, X is initialized arbitrarily as $X = \{x_1, \ldots, x_k\}$. Empty buckets $B_1, \ldots, B_k$ are also initialized, corresponding to clusters that would eventually be served by $x_1, \ldots, x_k$ respectively. We assume we know the cost $\texttt{opt}$ of the optimal solution[6]. While there exists a client violating the desired $(1+\varepsilon)$-approximation of $\texttt{opt}$ – i.e. while $\exists v \in C : d(v, X) > (1+\varepsilon)\texttt{opt}$ – the algorithm guesses the index $i \in [k]$ of the optimal cluster that serves v in $\texttt{OPT}$. The client v is then placed in the bucket B_i, and x_i is recomputed so that $\forall u \in B_i, d(u, x_i) \le \texttt{opt}$. If such an x_i cannot be obtained, then the algorithm asserts failure and restarts from the initialization; but with positive probability $1/k$ we don't fail (because with probability $1/k$ we guess i correctly). Now here is the key insight, that each bucket B_i can't grow too large. Indeed, consider the sequence of requests $(v_i^{(1)}, \ldots, v_k^{(\ell)})$ added to B_i over time, and consider the sequence of candidate locations for x_i over time, $(x_i^{(1)}, \ldots, x_i^{(\ell)})$. These two sequences form an ε-scatter (after we normalize distances so $\texttt{opt} = 1$). In particular, if the ε-scatter dimension of d is upper-bounded by some $\lambda(\varepsilon)$, then every bucket B_i only grows to size $\lambda(\varepsilon)$ until we either assert success or failure. As there there are k buckets overall, each iteration of the algorithm requires at most $k\lambda(\varepsilon)$ random choices each of which is correct with probability $1/k$; so with constant probability, we successfully find a solution after $O_{\varepsilon,k}(1)$ re-initializations.

For our aggregate clustering problem, we generalize the above. Rather than guessing a single partition of the clients into clusters served by $x_1, \ldots, x_k$, we need to guess T different partitions, one for each metric: we now have buckets $B_{i,t}$ for each $i \in [k], t \in [T]$, and we separately guess $\texttt{opt}_t = \text{cost}(d_t; \texttt{OPT})$ for each t. These metric-wise buckets allows us to mostly decouple the different metrics: when we find a violation in metric d_t, i.e. get $\text{cost}(d_t; X) > (1 + \Theta(\varepsilon))\texttt{opt}_t$, we select a random index $i \in [k]$ and add to bucket $B_{i,t}$, just as is done for the $T = 1$ case. Only when recomputing x_i do we take the other metrics

[6] We can approximate it by bucketing by powers of $1+\varepsilon$ and guessing over the $O(\varepsilon^{-1} \log n)$ options.

into account; we choose x_i to obey the constraints of all the buckets $B_{i,t}$ across all $t \in [T]$, that is $\forall t \in [T] \forall u \in B_{i,t}, d_t(u, x_i) \leq \mathtt{opt}_t$. Observe that each bucket $B_{i,t}$ still has bounded size, as it provides witness of an ε-scatter in metric d_t. To be precise, if one considers the sequence of requests $(v_i^{(1)}, \ldots, v_i^{(\ell)})$ in $B_{i,t}$ and the sequence of candidate positions for x_i *at the moment immediately before request* v_i^j *was added*[7], $(x_i^{(1)}, \ldots, x_i^{(\ell)})$, then these sequences form an ε-scatter for d_t. In particular, each iteration of the algorithm makes $kT\lambda(\varepsilon)$ random choices, and so it succeeds with constant probability after $O_{\varepsilon,k,T}(1)$ rounds.

▶ **Theorem 21.** *When all T metrics have ε-scatter dimension at most $\lambda(\varepsilon)$, for any homogeneous aggregator Ψ there is an algorithm for Ψ-aggregate (unweighted) k-supplier runs in time $O_{\varepsilon,k,T}(1) \cdot \mathrm{poly}(n)$, and produces a $(1+\varepsilon)$-approximation with constant probability.*

When $T = 1$, the k-supplier EPAS generalizes [1] to weighted k-center (by initializing with the help of a $O(1)$-approximation), then to weighted k-median (by a random sampling argument), and then to more general k-clustering problems. We adapt some of these ideas to the aggregate setting, and prove the following theorem in the full version [16].

▶ **Theorem 22.** *When all T metrics have ε-scatter dimensions upper-bounded by some $\lambda(\varepsilon)$, for any homogeneous aggregator Ψ there is an algorithm for Ψ-aggregate k-median that runs in time $O_{\varepsilon,k,T}(1) \cdot \mathrm{poly}(n)$, and produces $(1+\varepsilon)$-approximations with constant probability.*

▶ Remark 23 (Barrier at algorithmic scatter dimension). The authors of [1] also consider a weaker notion of *algorithmic ε-scatter dimension* (where, roughly speaking, the length of ε-scatters are only bounded if the sequence of centers x_i is chosen according to a certain algorithm), and their EPASes extend to metrics of bounded algorithmic scatter dimension. For example in the case of k-supplier, when recomputing the center c_i for some bucket B_i, they just use the algorithm for computing the center rather than choosing x_i to be an arbitrary point that covers B_i. We are aware of two follow-up works [38, 12] that build on the framework of [1], both of which apply also for algorithmic scatter dimension. In contrast, our aggregate algorithm *does not extend* to the more general setting; this is because we recompute x_i in a way that covers all the buckets $B_{i,t}$ across $t \in [T]$ (and such an x_i is guaranteed to exist, if we guessed the buckets correctly), whereas an algorithmic version would require that x_i was chosen to simultaneously satisfy the algorithms of all T metrics (and such an x_i does *not* necessarily exist).

5.2 Bounded-treewidth Graphs

▶ **Theorem 24.** *When the base graph G has treewidth tw and the aggregator $\Psi = sum$, then:*

- *there is an exact algorithm for sum-aggregate (k,z)-clustering that runs in time $n^{O(T \cdot \mathsf{tw})}$.*
- *for any $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm for sum-aggregate (k,z)-clustering that runs in time $(\frac{\log n}{\varepsilon})^{O(T \cdot \mathsf{tw})} \cdot \mathrm{poly}(n) = O_{\varepsilon,T,\mathsf{tw}}(1) \cdot \mathrm{poly}(n)$.*

In this section we focus on the k-median case; the ideas extend naturally to (k,z)-clustering and k-supplier. We prove Theorem 24 by adapting a folklore dynamic program for solving k-median on bounded-treewidth graphs. (Recall that a tree decomposition of a graph G is a tree $\mathcal{T}$ of *bags*, where each bag is a subset of vertices $V(G)$, that satisfies two properties:

[7] We remark that, unlike the $T = 1$ case, the center x_i could take on more than ℓ candidate positions, because the the position of x_i could be changed due to a request in any bucket $B_{i,t'}$ for $t' \in [T]$; nevertheless, even if x_i is changed due to a different bucket $B_{i,t'}$ we still guarantee that x_i covers requests in $B_{i,t}$, and so one can construct an ε-scatter.

(1) for every edge (u, v) in G, there is a bag containing both u and v, and (2) for every vertex v in $V(G)$, the bags containing v induce a connected subtree of $\mathcal{T}$. The *width* of the tree decomposition is 1 less than the maximum bag size, and the *treewidth* of a graph is the minimum width of any tree decoposition. A tree has treewidth 1.) To the best of our knowledge, the DP for k-median has not been written down explicitly in the setting of bounded treewidth graphs, but the technique is standard. Our presentation is based on a DP that appeared [6, 42, 29] as part of a PTAS for k-median in Euclidean metrics. In this section we explain the main idea of the algorithm and defer the formal details to the full version [16].

We begin by reviewing the folklore DP. Every bag B of the tree decomposition $\mathcal{T}$ acts as a small separator between an "inside" piece (induced by the vertices in the subtree of $\mathcal{T}$ rooted at B) and an "outside" piece (induced by all vertices outside the subtree rooted at B). For each separator, the DP guesses the *interface* of the optimal solution $\texttt{OPT}$ between the two pieces. In the case of k-median, the interface of bag $B = \{v_1, \ldots, v_{\mathsf{tw}}\}$ consists of the distance of every v_i to the closest open center in the inside (resp. outside) piece, as well as the number of centers opened in the inside (resp. outside) piece in the optimal solution. Note that there are $n^{O(\mathsf{tw})}$ possible interfaces for any bag, as there are $O(n^2)$ possible distances for each v_i; by rounding distances to the nearest power of $(1 + \frac{\varepsilon}{\log n})$, we could reduce the number of interfaces to $(\frac{\log n}{\varepsilon})^{O(\mathsf{tw})}$. The DP table contains a cell for every bag B and every possible interface of B, where the value of the cell represents the cost paid by vertices in the inside piece under the best solution that respects the interface. The value of a DP cell at bag B can be computed from the DP cells of the children of B, by taking a minimum over all combinations of children interfaces which are *consistent* with the interface of B (checking the consistency of interfaces can be done in a straightforward manner in polynomial time [6, 29]). The overall runtime is $n^{O(\mathsf{tw})}$, or $(\frac{\log n}{\varepsilon})^{O(\mathsf{tw})} \cdot \mathrm{poly}(n)$ if we tolerate $1 + \varepsilon$ approximation.

To adapt the DP to the aggregate setting, we simply change the notion of *interface*: rather than guessing just one interface, we guess a different interface for each of the T metrics $d_1, \ldots, d_T$. The cells of our DP table now consist of a bag $B = \{v_1, \ldots, v_{\mathsf{tw}}\}$ and a set of T tuples; the t-th tuple consists of the distance *with respect to* d_t of every v_i to the closest (with respect to d_t) open center in $\texttt{OPT}$ in each piece, as well as the cost paid by each piece in the d_t metric under $\texttt{OPT}$. (Note: the reason that we need to guess a different interface for each metric is that, even though the optimal set of centers $\texttt{OPT}$ for the aggregate problem stays consistent throughout the T metrics, a vertex v could be served by a different center of $\texttt{OPT}$ in each different metric d_t.) There are $n^{O(T \cdot \mathsf{tw})}$ possible interfaces. The DP can be computed in a bottom-up fashion, as in the non-aggregate setting, for a total runtime of $n^{O(T \cdot \mathsf{tw})}$. If one allows $1 + \varepsilon$ approximation, the runtime improves to $(\frac{\log n}{\varepsilon})^{O(T \cdot \mathsf{tw})} \cdot \mathrm{poly}(n)$, which is at most $\varepsilon^{-O(T^2 \cdot \mathsf{tw}^2)} \cdot \mathrm{poly}(n) = O_{\varepsilon,T,\mathsf{tw}}(1) \cdot \mathrm{poly}(n)$ (see Fact 26).

▶ Remark 25. Observe that if $\mathsf{tw} = o\left(\frac{\log n}{\log \log n}\right)$, then Theorem 24 provides a $(1 + \varepsilon)$-approximation algorithm in $2^{o(\log n) \cdot T \cdot \log(1/\varepsilon)} \cdot \mathrm{poly}(n) = O_{\varepsilon,T}(1) \cdot \mathrm{poly}(n)$ time (by Fact 26).

References

1 Fateme Abbasi, Sandip Banerjee, Jarosław Byrka, Parinya Chalermsook, Ameet Gadekar, Kamyar Khodamoradi, Dániel Marx, Roohani Sharma, and Joachim Spoerhase. Parameterized approximation schemes for clustering with general norm objectives. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1377–1399. IEEE, 2023. doi:10.1109/FOCS57990.2023.00085.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension and provably efficient shortest path algorithms. *Journal of the ACM (JACM)*, 63(5):1–26, 2016. doi:10.1145/2985473.

3 Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 782–793. SIAM, 2010.

4 Akanksha Agrawal, Tanmay Inamdar, Saket Saurabh, and Jie Xue. Clustering what matters: Optimal approximation for clustering with outliers. *J. Artif. Int. Res.*, 78, 2024.

5 Barbara Anthony, Vineet Goyal, Anupam Gupta, and Viswanath Nagarajan. A plant location guide for the unsure: Approximation algorithms for min-max location problems. *Mathematics of Operations Research*, 35(1):79–101, 2010. Preliminary version appeared in SODA 2008. doi:10.1287/MOOR.1090.0428.

6 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, 1998. doi:10.1145/276698.276718.

7 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 (SICOMP)*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

8 MohammadHossein Bateni, Hossein Esfandiari, Hendrik Fichtenberger, Monika Henzinger, Rajesh Jayaram, Vahab Mirrokni, and Andreas Wiese. Optimal fully dynamic k-center clustering for adaptive and oblivious adversaries. In *the Symposium on Discrete Algorithms (SODA)*, pages 2677–2727, 2023. doi:10.1137/1.9781611977554.CH101.

9 Sayan Bhattacharya, Martín Costa, and Ermiya Farokhnejad. Fully dynamic k-median with near-optimal update time and recourse. In *the Symposium on Theory of Computing (STOC)*, pages 1166–1177, 2025. doi:10.1145/3717823.3718176.

10 Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in $\tilde{O}(k$ update time. *Adv. in Neural Infomation Processing Systems (NeurIPS)*, 36:18633–18658, 2023.

11 Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-center clustering made simple. In *the International Conference on Machine Learning (ICML)*, 2025.

12 Sujoy Bhore, Ameet Gadekar, and Tanmay Inamdar. Coreset strikes back: Improved parameterized approximation schemes for (constrained) k-median/means. *arXiv preprint arXiv:2504.06980*, 2025. doi:10.48550/arXiv.2504.06980.

13 Romain Bourneuf and Marcin Pilipczuk. Bounding ε-scatter dimension via metric sparsity. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3155–3171. SIAM, 2025. doi:10.1137/1.9781611978322.101.

14 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *the Symposium on Discrete Algorithms (SODA)*, pages 737–756, 2014.

15 Constantin Carathéodory. Über den variabilitätsbereich der fourier'schen konstanten von positiven harmonischen funktionen. *Rendiconti Del Circolo Matematico di Palermo (1884-1940)*, 32(1):193–217, 1911.

16 Deeparnab Chakrabarty, Jonathan Conroy, and Ankita Sarkar. Clustering in varying metrics, 2025. arXiv:2510.07860.

17 Deeparnab Chakrabarty, Luc Cote, and Ankita Sarkar. Fault-tolerant k-supplier with outliers. In *the Symposium on Theoretical Aspects of Computer Science (STACS)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.STACS.2024.23.

18 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The Non-Uniform k-Center Problem. *the ACM Transactions on Algorithms (TALG)*, 2020. Preliminary version in ICALP 2016. doi:10.1145/3392720.

19 Deeparnab Chakrabarty and Maryam Negahbani. Generalized Center Problems with Outliers. *the ACM Transactions on Algorithms (TALG)*, 2019. Prelim. version in ICALP 2018. doi:10.1145/3338513.

20 Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar. Approximation algorithms for continuous clustering and facility location problems. In *European Symposium on Algorithms*, 2022.

21 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *the Symposium on Theory of Computing (STOC)*, pages 126–137, 2019. doi:10.1145/3313276.3316322.

22 TH Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *the International World Wide Web Conference*, pages 579–587, 2018.

23 Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002. Preliminary version appeared in STOC 1999. doi:10.1006/JCSS.2002.1882.

24 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *the Symposium on Discrete Algorithms (SODA)*, volume 1, pages 642–651. Citeseer, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365555.

25 Moses Charikar and Shi Li. A dependent lp-rounding approach for the k-median problem. In *International Colloquium on Automata, Languages, and Programming*, pages 194–205. Springer, 2012. doi:10.1007/978-3-642-31594-7_17.

26 Eden Chlamtáč, Yury Makarychev, and Ali Vakilian. Approximating fair clustering with cascaded norm objectives. In *the Symposium on Discrete Algorithms (SODA)*, 2022.

27 Fabián A Chudak and David B Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *Journal of the ACM (JACM)*, 33(1):1–25, 2003. doi:10.1137/S0097539703405754.

28 Julia Chuzhoy and Zihan Tan. Towards tight (er) bounds for the excluded grid theorem. *Journal of Combinatorial Theory, Series B*, 146:219–265, 2021. doi:10.1016/J.JCTB.2020.09.010.

29 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *Journal of the ACM (JACM)*, 68(6):1–34, 2021. doi:10.1145/3477541.

30 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight fpt approximations for k-median and k-means. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 42–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.42.

31 Vincent Cohen-Addad, Niklas Oskar D Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. *Adv. in Neural Infomation Processing Systems (NeurIPS)*, 32, 2019.

32 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

33 Shichuan Deng, Jian Li, and Yuval Rabani. Approximation algorithms for clustering with dynamic points. *Journal of Computer and System Sciences*, 130:43–70, 2022. doi:10.1016/J.JCSS.2022.07.001.

34 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

35 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011. doi:10.1145/1993636.1993712.

36 Hendrik Fichtenberger, Silvio Lattanzi, Ashkan Norouzi-Fard, and Ola Svensson. Consistent k-clustering for general metrics. In *the Symposium on Discrete Algorithms (SODA)*, pages 2660–2678, 2021. doi:10.1137/1.9781611976465.158.

37 Sebastian Forster and Antonis Skarlatos. Dynamic consistent k-center clustering with optimal recourse. In *the Symposium on Discrete Algorithms (SODA)*, pages 212–254, 2025. doi:10.1137/1.9781611978322.7.

38 Ameet Gadekar and Tanmay Inamdar. Dimension-Free Parameterized Approximation Schemes for Hybrid Clustering. In Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyên Kim Thang, editors, *42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)*, volume 327 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2025.35.

39 Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

40 AJ Hoffman and JB Kruskal. Integral boundary points of convex polyhedra, in "linear inequalities and related systems"(hw kuhn and aw tucker, eds.). *Annals of Mathematical Studies*, 38, 1956.

41 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangean relaxation. *Journal of the ACM (JACM)*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

42 Stavros G Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM Journal on Computing*, 37(3):757–782, 2007. doi:10.1137/S0097539702404055.

43 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *the Symposium on Theory of Computing (STOC)*, pages 646–659, 2018. doi:10.1145/3188745.3188882.

44 Shrinu Kushagra. Three-dimensional matching is np-hard. *arXiv preprint arXiv:2003.00336*, 2020. arXiv:2003.00336.

45 Shrinu Kushagra, Shai Ben-David, and Ihab Ilyas. Semi-supervised clustering for de-duplication. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1659–1667. PMLR, 2019. URL: http://proceedings.mlr.press/v89/kushagra19a.html.

46 Jakub Łącki, Bernhard Haeupler, Christoph Grunau, Rajesh Jayaram, and Václav Rozhoň. Fully dynamic consistent k-center clustering. In *the Symposium on Discrete Algorithms (SODA)*, pages 3463–3484, 2024.

47 Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *the International Conference on Machine Learning (ICML)*, pages 1975–1984, 2017. URL: http://proceedings.mlr.press/v70/lattanzi17a.html.

48 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013. doi:10.1016/J.IC.2012.01.007.

49 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing (SICOMP)*, 45(2):530–547, 2016. doi:10.1137/130938645.

50 Sepideh Mahabadi and Ali Vakilian. (Individual) Fairness for k-Clustering. In *the International Conference on Machine Learning (ICML)*, pages 7925–7935, 2020. URL: https://dl.acm.org/doi/abs/10.5555/3524938.3525549.

51 Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *the Symposium on Theory of Computing (STOC)*, pages 1027–1038, 2019. doi:10.1145/3313276.3316350.

52 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In *the Conference on Learning Theory (COLT)*, 2021.

53 Ján Plesník. A heuristic for the p-center problems in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987. doi:10.1016/0166-218X(87)90029-1.

54 Neil Robertson and Paul D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

55 David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *the Symposium on Theory of Computing (STOC)*, pages 265–274, 1997.

56 Ernst Steinitz. Bedingt konvergente reihen und konvexe systeme. *Journal für die reine und angewandte Mathematik*, 1913.

57 Chaitanya Swamy. Improved approximation algorithms for matroid and knapsack median problems and applications. *the ACM Transactions on Algorithms (TALG)*, 12(4):1–22, 2016. doi:10.1145/2963170.

58 Li Yan and Marek Chrobak. Lp-rounding algorithms for the fault-tolerant facility placement problem. *Journal of Discrete Algorithms*, 33:93–114, 2015. doi:10.1016/J.JDA.2015.03.004.

59 Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *the International Conference on Machine Learning (ICML)*, pages 928–936, 2003. URL: http://www.aaai.org/Library/ICML/2003/icml03-120.php.

A Useful facts

▶ **Fact 26.** *For any $f(n) = n^{o(1)}$ and any variable $\tau \geq 1$, the function $(f(n))^\tau$ is bounded above by $O_\tau(1) + n$. For example, when $f(n) = \log n$, we have $(\log n)^\tau \leq 2^{\tau^2} + n$.*

Proof. Let $g(n) = \log f(n)$, and note $g(n) = o(\log n)$. If τ satisfies $k\tau \leq \log n$, then clearly $f(n)^\tau \leq n$. On the other hand, if $\tau > \frac{\log n}{k}$, then the fact that $g(n) = o(\log n)$ implies that τ grows as a function of n, and so $f(n)^\tau = O_\tau(1)$.

We now give an example that provides explicit bounds for the case $f(n) = \log n$. If $\tau \leq \log\log n$, then $(\log n)^\tau \leq 2^{(\log\log n)^2} \leq n$. Otherwise, if $\tau > \log\log n$, then $(\log n)^\tau = 2^{\tau \log\log n} \leq 2^{\tau^2}$. ◀

▶ **Observation 27.** *Fix an aggregator function Ψ. Suppose that one can find an α-approximation to the Ψ-aggregate k-supplier (or k-median, or (k,z)-clustering) problem on T metrics and n vertices in time $f(n,T,k,\alpha)$. Then one can find an α-approximation to generalized aggregate clustering, with the restriction that weights come from $\{0,1\}$, in time $f(2n+1,T,k+1,\alpha) + \mathrm{poly}(n,T)$.*

Proof. Suppose we are given an instance η of $\{0,1\}$-weighted generalized aggregate k-clustering, defined by metrics $d_1,\ldots,d_T$, client set C, and facility set F. We construct an instance η' of vanilla aggregate $(k+1)$-clustering as follows. Create a new vertex r, and let the new facility set F' be $F' := F \cup \{r\}$. The client set C' is set to be C. We assume WLOG that C' and F' are disjoint: if some vertex v is both a client and facility, we duplicate v, and make one copy a client and one a facility. After this duplication, we have $|F' \cup C'| \leq 2|F \cup C| + 1 = 2n + 1$.

For every $t \in [T]$, we construct a new distance metric d'_t as follows. We define the *active vertices* to be the set of clients with weight 1 in the t-th instance, together with the set of facilities excluding r (that is, $F' \setminus \{r\} = F$). The *inactive vertices* are those clients with weight 0 in the t-th instance, and the facility r. For any two vertices $v_1, v_2 \in C \cup F$, we define

$$\hat{d}'_t(v_1,v_2) := \begin{cases} d_t(v_1,v_2) & \text{if } v_1 \text{ and } v_2 \text{ are both active} \\ \infty & \text{if one of } \{v_1,v_2\} \text{ is active and the other is inactive} \\ 0 & \text{if } v_1 \text{ and } v_2 \text{ are both inactive} \end{cases}$$

One can compute a solution to the instance η from a solution to η'. Indeed, suppose $S' \subseteq F'$ is a set of $k+1$ facilities with aggregate cost κ in η'. It is straightforward to see that, if κ' is finite, then facility r is included in S', and the set $S := S' \setminus \{r\}$ has aggregate

cost κ for the instance η. Moreover, if $S \subseteq F$ is any set of k facilities with aggregate cost κ in η, then the set of facilities $S' \coloneqq S \cup \{r\}$ has aggregate cost κ in η'. This proves the claim: in polynomial time one can construct η', and in time $f(2n+1, T, k+1, \alpha)$ one can compute an approximate solution S' and return $S' \setminus \{r\}$. ◀

Flavors of Quantifiers in Hyperlogics

Marek Chalupa
IST Austria, Klosterneuburg, Austria

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria

Ana Oliveira da Costa
IST Austria, Klosterneuburg, Austria

Abstract

Hypertrace logic is a sorted first-order logic with separate sorts for time and execution traces. Its formulas specify hyperproperties, which are properties relating multiple traces. In this work, we extend hypertrace logic by introducing trace quantifiers that range over the set of all possible traces. In this extended logic, formulas can quantify over two kinds of trace variables: *constrained trace variables*, which range over a fixed set of traces defined by the model, and *unconstrained trace variables*, which can be assigned to any trace. In comparison, hyperlogics such as HyperLTL have only constrained trace quantifiers. We use hypertrace logic to study how different quantifier patterns affect the decidability of the satisfiability problem. We prove that hypertrace logic without constrained trace quantifiers is equivalent to monadic second-order logic of one successor (S1S), and therefore satisfiable, and that the trace-prefixed fragment (all trace quantifiers precede all time quantifiers) is equivalent to HyperQPTL. Moreover, we show that all hypertrace formulas where the only alternation between constrained trace quantifiers is from an existential to a universal quantifier are equisatisfiable to formulas without constraints on their trace variables and, therefore, decidable as well. Our framework allows us to study also time-prefixed hyperlogics, for which we provide new decidability and undecidability results.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Hyperproperties, Satisfiability, First-order Logic, S1S

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.20

Related Version *Full Version*: https://arxiv.org/abs/2510.12298

Funding This work was supported in part by the Austrian Science Fund (FWF) SFB project SpyCoDe 10.55776/F85 and by the ERC Advanced Grant VAMOS 101020093.

1 Introduction

Temporal logics offer a powerful formalism for specifying *trace properties*, which describe sets of allowed sequences of events (or system states) that a system can exhibit over time. Within the linear-time spectrum, Linear Temporal Logic (LTL) has been particularly successful in system verification, striking a practical balance between expressiveness and decidability. Its extension with propositional quantifiers, known as Quantified Propositional Temporal Logic (QPTL), extends LTL's expressive power to capture all regular trace properties while preserving decidability. The connection between temporal logics and classical first-order logic for specifying linear-time properties was first explored in the seminal work of Kamp [12]. This line of research established that LTL is expressively equivalent to first-order logic of order (FO[<]), where the order relation $<$ is interpreted over the natural numbers. In the case of QPTL, it was proven to be expressively equivalent to monadic second-order logic of order of one successor (S1S) [13]. However, these logics are inherently limited to reasoning about individual execution traces and cannot express properties that involve comparing multiple executions. In particular, they are not capable of capturing *hyperproperties*.


© Marek Chalupa, Thomas A. Henzinger, and Ana Oliveira da Costa;
licensed under Creative Commons License CC-BY 4.0



To address this limitation of trace-based formalisms in expressing hyperproperties, numerous extensions have been proposed in the literature. Notably, HyperLTL and HyperQPTL extend LTL and QPTL, respectively, with quantification over traces of the system under verification. An alternative approach involves extending FO[<] with capabilities to compare multiple traces. *Hypertrace logic*, introduced in [1], extends FO[<] into a two-sorted first-order logic, with separate sorts for time and traces of a given system, allowing only binary predicates over traces and time.

In this work, we propose an extension of hypertrace logic that includes both *constrained trace quantifiers*, ranging over all traces in the set given as a model, and *unconstrained trace quantifiers*, ranging over the universe of all traces. Properties with a mix of constrained and unconstrained trace quantification occur naturally in many areas of system verification. For example, in reactive systems, *input enableness* requires that *"the system must produce an output for any possible input"* which can be specified as:

$$\forall \pi \; \exists \pi' :: T \; \forall i \; \big((\mathrm{input}(\pi, i) \leftrightarrow \mathrm{input}(\pi', i)) \wedge \mathrm{outputs}(\pi', i) \big). \tag{1}$$

By using the unconstrained quantifier, $\forall \pi$, we ensure that all possible input values are taken into account, not just those accepted by the reactive system. By linking the universally quantified inputs to those instantiated by the constrained existential quantifier, $\exists \pi' :: T$, we guarantee that every possible input is in at least one system execution.

Another example, can be found in the specification of consistency properties in concurrent systems like *linearizability* [11], which requires all histories of invocations and response events to shared resources to be consistent with some sequential execution of those operations. While verifying for linearizability, it is irrelevant whether the implementation of a concurrent data structure exhibits linear histories because all that matters is that its histories can be successfully related to the "idealized" sequential implementation. In its essence, linearizability can be captured by the following hyperproperty:

$$\forall \pi :: T \; \exists \pi' \; (\varphi_{\mathrm{linear}}(\pi') \wedge \varphi_{\mathrm{equivOrder}}(\pi, \pi')). \tag{2}$$

The formula above universally quantifies over the observed call history of the shared object while requiring the existence of a linear trace ($\varphi_{\mathrm{linear}}(\pi')$) that is equivalent to the observed one and respects the precedence order defined between the observed ($\varphi_{\mathrm{equivOrder}}(\pi, \pi')$).

We prove that the fragment of hypertrace logic without constrained trace quantifiers, named *unconstrained hypertrace logic*, is expressively equivalent to S1S. It follows directly that unconstrained hypertrace logic has a decidable satisfiability problem. We show how to translate hypertrace formulas whose only constrained trace quantifier alternation is from existential to universal quantification into an equisatisfiable unconstrained hypertrace formula. Hence, this fragment also has a decidable satisfiability problem. This fragment places no limitations on the positioning of unconstrained or temporal quantifiers after the constrained quantification. For the fragment where all trace quantifiers occur before time quantifiers, we prove it to be expressively equivalent to HyperQPTL. While the majority of satisfiability results for hyperlogics in the literature focus on the trace-prefix fragment, we extend this line of work by also analyzing the fragment of hypertrace logic where time quantifiers occur first. For the time-prefix fragment, we prove a new undecidability result by presenting a reduction from the non-halting problem for 2-counter Minsky machines. Our main contribution is in highlighting the role of different quantifiers, including unconstrained trace quantification, plays in defining hyperlogics with a decidable satisfiability problem.

2 Logics of Time and Order

In this section, we present the necessary theoretical background, including definitions and prior results, which will form the basis for the results introduced in the next sections.

Let $\mathcal{X}$ be a finite set of boolean variables. A *valuation* is a partial mapping assigning boolean values, $\mathbf{2}=\{0,1\}$, to variables in $\mathcal{X}$, that is, $v:\mathcal{X}\to\mathbf{2}$. We denote by $v[x\mapsto b]$ the valuation resulting from updating the value of x in v to b, and the *set of all valuations of* $\mathcal{X}$ by $\mathbf{2}^{\mathcal{X}}$. We freely treat a valuation as a set of variables where suitable. In particular, we write $x\in v$ when $v(x)=1$. A *trace* τ *over variables* $\mathcal{X}$ is a sequence of valuations in $\mathbf{2}^{\mathcal{X}}$. The set of all *finite* traces over $\mathcal{X}$ is denoted $(\mathbf{2}^{\mathcal{X}})^*$, while the set of all *infinite* traces over $\mathcal{X}$ is denoted by $(\mathbf{2}^{\mathcal{X}})^\omega$. For a trace $\tau=v_0v_1\dots$ and an index i within its length (i.e., $i<|\tau|$), we adopt the following indexing notations: $\tau[i]=v_i$, $\tau[i\dots]=v_iv_{i+1}\dots$, and $\tau[\dots i]=v_0v_1\dots v_{i-1}$. When an index falls outside a trace length (i.e., $j\geq|\tau|$), we adopt the convention that $\tau[j\dots]$ is the empty trace, and $\tau[\dots j]=\tau$. Two traces, τ and τ', agree on their valuation of a set of propositional variables $\mathcal{Y}$ at position i, denoted $\tau[i]=_{\mathcal{Y}}\tau'[i]$, iff $(\tau[i]\cap\mathcal{Y})=(\tau'[i]\cap\mathcal{Y})$. We generalize it to equality between two traces of equal size with respect to a given set of variables $\mathcal{Y}$, denoted $\tau=_{\mathcal{Y}}\tau'$, when, for all $i<|\tau|$, $\tau[i]=_{\mathcal{Y}}\tau'[i]$.

Trace properties define set of traces satisfying a target specification. Formally, a trace property T over $\mathcal{X}$ is a set of traces over $\mathcal{X}$; for infinite traces, that is, $\mathrm{T}\subseteq(\mathbf{2}^{\mathcal{X}})^\omega$. A trace τ satisfies a trace property T iff it is one of its elements, that is $\tau\in\mathrm{T}$. We refer to the set of all trace properties as $\mathbb{T}=\mathbf{2}^{(\mathbf{2}^{\mathcal{X}})^\omega}$. For systems represented by a set of traces S, where each trace corresponds to one of the system's execution, the system is said to satisfy the trace property T if and only if all its traces are in T; that is, $S\subseteq\mathrm{T}$. A *hyperproperty* $\mathbf{T}$ defines a set of systems (or, equivalently a set of trace properties) satisfying a given specification, $\mathbf{T}\subseteq\mathbb{T}$. Hence, a system S satisfies a hyperproperty $\mathbf{T}$ if and only if $S\in\mathbf{T}$.

2.1 Linear Temporal Logics

A successful formalism to specify trace properties is Linear Temporal Logic (LTL), introduced by Pnueli in [16]. LTL formulas are defined by the grammar: $\varphi ::= a\mid\neg\varphi\mid\varphi\vee\varphi\mid\mathbf{X}\varphi\mid\varphi\,\mathbf{U}\,\varphi$ where $a\in\mathcal{X}$ is a propositional variable, and $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until") are temporal modalities. Given a LTL formula φ and an infinite trace $\tau\in(\mathbf{2}^{\mathcal{X}})^\omega$, the trace τ satisfies φ, denoted $\tau\models_{\mathtt{LTL}}\varphi$, inductively over the structure of φ, as follows:

$\tau\models_{\mathtt{LTL}}a$ iff $a\in\tau[0]$ $\qquad$ $\tau\models_{\mathtt{LTL}}\neg\psi$ iff $\tau\not\models_{\mathtt{LTL}}\psi$

$\tau\models_{\mathtt{LTL}}\psi_1\vee\psi_2$ iff $\tau\models_{\mathtt{LTL}}\psi_1$ or $\tau\models_{\mathtt{LTL}}\psi_2$ $\qquad$ $\tau\models_{\mathtt{LTL}}\mathbf{X}\psi$ iff $\tau[1\dots]\models_{\mathtt{LTL}}\psi$

$\tau\models_{\mathtt{LTL}}\psi_1\,\mathbf{U}\,\psi_2$ iff exists $j\geq0$ s.t. $\tau[j\dots]\models_{\mathtt{LTL}}\psi_2$ and for all $0\leq j'<j,\tau[j'\dots]\models_{\mathtt{LTL}}\psi_1$.

QPTL extends LTL with propositional quantification. Concretely, QPTL formulas may include subformulas with propositional quantifiers (e.g., $\exists q\ \varphi$, where $q\in\mathcal{X}$ and φ is a QPTL formula) interpreted as: $\tau\models_{\mathtt{QPTL}}\exists q\ \psi$ iff exists τ' s.t. $\tau=_{\mathcal{X}\setminus\{q\}}\tau'$ and $\tau'\models_{\mathtt{QPTL}}\psi$. All other subformulas are interpreted as for LTL.

2.2 Monadic Logics of Order

Monadic second-order logic of one successor (S1S) is a widely used formalism for reasoning about regular properties of infinite sequences. S1S formulas φ are defined by the grammar:

$$\varphi ::= \exists X\ \varphi \mid \exists i\ \varphi \mid \neg\varphi \mid \varphi\vee\varphi \mid i=i \mid \mathrm{Succ}(i,i) \mid X(i) \tag{3}$$

where X is a second-order variable from a set of variables $\mathcal{V}_2$, i is a first-order variable over a set of variables $\mathcal{V}_1$ and Succ is a successor function. The set of first and second order variables are disjoint; that is, $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$. Unless specified otherwise, we use uppercase letters for second-order variables, $\mathcal{V}_2 = \{X, Y, \ldots\}$, and lowercase letter for first-order variables, $\mathcal{V}_1 = \{i, j, \ldots\}$. The set of free variables of a formula (i.e., not bounded to a quantifier) and closed formulas (no free variables) are defined as usual.

We interpret S1S formulas over the natural numbers, where the $\text{Succ}(n, n')$ is interpreted as usual: $\text{Succ}(n, n')$ iff $n' = n + 1$. We adopt the following abbreviations:

- $X \subseteq Y \stackrel{\text{def}}{=} \forall i \ (X(i) \rightarrow Y(i))$;
- $\text{SuccClosed}(X) \stackrel{\text{def}}{=} \forall i \ \forall i' \ ((X(i) \wedge \text{Succ}(i, i')) \rightarrow X(i'))$;
- $x \leq y \stackrel{\text{def}}{=} \forall Z((Z(x) \wedge \text{SuccClosed}(Z)) \rightarrow Z(y))$;
- $x < y \stackrel{\text{def}}{=} x \leq y \wedge \neg(x = y)$;
- $i = 0 \stackrel{\text{def}}{=} \forall j \ (i = j \vee i < j)$.

The semantics of S1S formulas is defined inductively with respect to two assignments: one for first-order variables, $\Pi_1 : \mathcal{V}_1 \mapsto \mathbb{N}$, and another for second-order variables, $\Pi_2 : \mathcal{V}_2 \mapsto \mathbf{2}^{\mathbb{N}}$. For any assignment Π, $\Pi[x \mapsto v]$ denotes the assignment that is the same as Π except for the value of x that is updated to v. We define that (Π_1, Π_2) satisfies φ inductively, as follows:

$(\Pi_1, \Pi_2) \models_{\text{S1S}} \exists X \ \varphi$ iff exists $S \subseteq \mathbb{N}$ s.t. $(\Pi_1, \Pi_2[X \mapsto S]) \models_{\text{S1S}} \varphi$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} \exists i \ \varphi$ iff exists $k \in \mathbb{N}$ s.t. $(\Pi_1[i \mapsto k], \Pi_2) \models_{\text{S1S}} \varphi$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} \neg\varphi$ iff $(\Pi_1, \Pi_2) \not\models_{\text{S1S}} \varphi$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} \varphi \vee \varphi'$ iff $(\Pi_1, \Pi_2) \models_{\text{S1S}} \varphi$ or $(\Pi_1, \Pi_2) \models_{\text{S1S}} \varphi'$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} i = j$ iff $\Pi_1(i) = \Pi_1(j)$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} \text{Succ}(i, i')$ iff $\Pi_1(i') = \Pi_1(i) + 1$

$(\Pi_1, \Pi_2) \models_{\text{S1S}} X(i)$ iff $\Pi_1(i) \in \Pi_2(X)$

S1S formulas define a set with all their satisfying assignments:
$[\![\varphi]\!] = \{(\Pi_1, \Pi_2) \mid (\Pi_1, \Pi_2) \models_{\text{S1S}} \varphi\}$.

▶ **Theorem 1** ([19]). *For all S1S formulas φ, it is decidable to determine whether $[\![\varphi]\!] \neq \emptyset$.*

In his seminal work [12], Kamp studied of relative expressiveness between the classical first-order approach to specify linear-time – the first-order logic of order, FO[<] – and LTL. The first-order logic FO[<] is interpreted over labeled linear orders with all uninterpreted predicates being unary. Formally, FO[<] formulas φ are defined by the grammar:

$$\varphi ::= \exists i \, \varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid i < i \mid i = i \mid X(i) \tag{4}$$

where i is a first-order variable, $=$ is equality, $<$ is the order, and X is predicate from a given set of monadic predicates $\mathcal{X}$.

We work with interpretations of FO[<] over labeled linear-orders defined by a tuple $(\Lambda, <, \mathcal{I})$ where $<$ defines a linear order over the domain Λ, and $\mathcal{I}$ is a function that associates to each predicate symbol a subset of elements of Λ. From a trace τ, defined over the set of propositional variables $\mathcal{X}$, we can derive the labeled linear-order $(\mathbb{N}, <, \mathcal{I}_\tau)$ over the set of unary predicates $\{X_a \mid a \in \mathcal{X}\}$, where $<$ is interpreted as usual for natural numbers and $\mathcal{I}_\tau(X_a) = \{j \mid a \in \tau[j]\}$. Using this interpretation, it is straightforward to translate LTL formulas to equivalent FO[<] formulas interpreted over $(\mathbb{N}, <)$. Hence, FO[<] subsumes LTL. The only question remaining is whether LTL subsumes FO[<] interpreted over $(\mathbb{N}, <)$. Kamp

proved in [12] that LTL with both past and future temporal operators is equivalent to FO[<] over Dedekind complete orders. Later, Gabbay et al. [9] proved that when considering only future operators, LTL is complete for FO[<] under the interpretation of $<$ over the natural numbers. For the remaining of the manuscript, we are only interested in the linear order over natural numbers.

▶ **Theorem 2** ([12, 9]). LTL *and FO*[<] *interpreted over* $(\mathbb{N}, <)$ *are equally expressive.*

3 Hypertrace Logic

In this section, we extend hypertrace logic, introduced in [1], to support *quantification over unconstrained trace variables*; that is, variables ranging over the set of all possible traces.

3.1 Syntax and Semantics

Hypertrace logic [1], denoted FO[$<, \mathbb{T}$], extends FO[<] with a time sort $\mathbb{N}_<$ and a trace sort $\mathbb{T}$. While FO[<] allows only unary uninterpreted predicates[1], in FO[$<, \mathbb{T}$] we allow binary uninterpreted predicates defined over pairs of a trace and a time variable. In [1], hypertrace logic is defined only over *constrained trace variables*: trace variables ranging over a fixed set of traces. Here we lift the implicit constraints on trace variables: we quantify instead over the set of all possible traces while enabling trace variables to be constrained by a unary predicate. Formally, in this work, hypertrace formulas φ are defined by the grammar:

$$\varphi ::= \exists \pi \ \varphi \mid \exists \pi{::}T \ \varphi \mid \exists i \ \varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid i < i \mid i = i \mid X(\pi, i) \tag{5}$$

where π is a trace variable from a set $\mathcal{V}_\mathbb{T}$, i is a time variable from a set $\mathcal{V}_\mathbb{N}$ disjoint from $\mathcal{V}_\mathbb{T}$, T is a unary predicate over trace variables and X is a binary predicate over pairs of trace and time variables from a finite set $\mathbb{X}$. We typically use lowercase letters for predicates in $\mathbb{X}$ to reflect their correspondence with propositional variables in traces. Without loss of generality, we assume that all variables are quantified only once. We refer to $\exists \pi{::}T$ as a *(existential) constrained trace quantifier* and we may refer to π as a *constrained trace variable*. We introduce the usual abbreviations: $\forall x \ \varphi \stackrel{\text{def}}{=} \neg(\exists x \ \neg\varphi)$, where $x \in \{i, \pi\}$; $\forall \pi{::}T \varphi \stackrel{\text{def}}{=} \neg(\exists \pi{::}T \ \neg\varphi)$; and $\varphi \wedge \varphi' \stackrel{\text{def}}{=} \neg(\neg\varphi \vee \neg\varphi')$. Finally, constrained trace quantifiers are abbreviations for the following first-order formulas: $\exists \pi{::}T \ \varphi \stackrel{\text{def}}{=} \exists \pi \ (T(\pi) \wedge \varphi)$ and $\forall \pi{::}T \ \varphi \stackrel{\text{def}}{=} \forall \pi \ (T(\pi) \rightarrow \varphi)$.

▶ **Example 3.** We define independence between a secret input and a public output as: $\varphi \stackrel{\text{def}}{=} \forall \pi{::}T \forall \pi'{::}T \exists \pi_\exists{::}T \ \forall i \ ((\text{secret}(\pi, i) \leftrightarrow \text{secret}(\pi_\exists, i)) \wedge (\text{pub}(\pi', i) \leftrightarrow \text{pub}(\pi_\exists, i)))$. We can mix constrained and unconstrained quantifiers to require that a system produces all combinations of visible public outputs with possible input secret values (i.e., a combination of input enableness with independence between secret inputs and public outputs): $\varphi_{\text{mix}} \stackrel{\text{def}}{=} \forall \pi \ \forall \pi'{::}T \ \exists \pi_\exists{::}T \ \forall i \ ((\text{secret}(\pi, i) \leftrightarrow \text{secret}(\pi_\exists, i)) \wedge (\text{pub}(\pi', i) \leftrightarrow \text{pub}(\pi_\exists, i)))$.

Our models of interest are sets of traces. As FO[$<, \mathbb{T}$] is a first-order logic, we start by defining how to translate a set of traces to a first-order structure. We translate each propositional variable $a \in \mathcal{X}$ to the binary predicate $X_a(\tau, k)$, asserting that "variable a is true in trace $\tau \in (\mathbf{2}^\mathcal{X})^\omega$ at time $k \in \mathbb{N}$". Formally, given a set T of traces, we translate T to a structure $\overline{\text{T}}$ with signature: $(\mathbb{N}, (\mathbf{2}^\mathcal{X})^\omega; \ < \subseteq \mathbb{N} \times \mathbb{N}, T \subseteq (\mathbf{2}^\mathcal{X})^\omega, (X_a \subseteq (\mathbf{2}^\mathcal{X})^\omega \times \mathbb{N})_{a \in \mathcal{X}})$

[1] The binary predicates for equality, $=$, and the linear order, $<$, are interpreted by $(\mathbb{N}, <)$.

where $\mathbb{N}$ and $(\mathbf{2}^{\mathcal{X}})^\omega$ are the time and trace sort domains, respectively. The predicate $<$ is interpreted as the usual order on natural numbers, the unary predicate T is true for all traces in the set set of traces used to generated the structure[2], while for all variables $a \in \mathcal{X}$: $X_a = \{(\tau, k) \mid a \in \tau[k]\}$.

We evaluate hypertrace formulas over pairs of assignments for traces and time: $(\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}) : (\mathcal{V}_{\mathbb{T}} \to (\mathbf{2}^{\mathcal{X}})^\omega) \times (\mathcal{V}_{\mathbb{N}} \to \mathbb{N})$. A set T of traces is a model of a hypertrace formula $\varphi \in \mathrm{FO}[<, \mathbb{T}]$, denoted $\mathrm{T} \models_{\mathbb{T}} \varphi$, iff $\overline{\mathrm{T}}$ models φ under the standard first-order semantics. Formally, $\mathrm{T} \models_{\mathbb{T}} \varphi$, iff there exists a pair of assignments $(\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})$ such that $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models \varphi$, where $\models$ is the standard first-order logic models relation. In particular, trace quantifiers are interpreted as:

$$(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \exists \pi \ \varphi \text{ iff exists } \tau \in (\mathbf{2}^{\mathcal{X}})^\omega \text{ s.t. } (\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}[\pi \mapsto \tau], \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \varphi$$
$$(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \exists \pi :: T \ \varphi \text{ iff exists } \tau \in \mathrm{T} \text{ s.t. } (\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}[\pi \mapsto \tau], \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \varphi.$$

Hypertrace formulas define a set with all the set of traces it satisfies: $[\![\varphi]\!] = \{ \mathrm{T} \mid \mathrm{T} \models_{\mathbb{T}} \varphi\}$. From now on, we may refer to X_a just as a, and omit the subscript $\mathbb{T}$ in $\models_{\mathbb{T}}$, when clear.

▶ **Example 4.** Looking back to our previous example, consider the sets of traces defined over the variables secret and pub, where with each valuation v is represented as a set: $\mathrm{T}_0 = \{\{\}^\omega\}$, $\mathrm{T}_1 = \{\{\text{secret}, \text{pub}\}^\omega\}$, and $\mathrm{T}_{0,1} = \{\{\}^\omega, \{\text{secret}, \text{pub}\}^\omega\}$. For the fully constrained formula in Example 3, $\mathrm{T}_0 \models_{\mathbb{T}} \varphi$ and $\mathrm{T}_1 \models_{\mathbb{T}} \varphi$ because in each of these traces we only observe one of the possible values for secret. While, $\mathrm{T}_{0,1} \not\models_{\mathbb{T}} \varphi$. For the mix formula, φ_{mix} none of sets satisfy its requirements; that is, $\mathrm{T}_0 \not\models_{\mathbb{T}} \varphi_{\text{mix}}$, $\mathrm{T}_1 \not\models_{\mathbb{T}} \varphi_{\text{mix}}$ and $\mathrm{T}_{0,1} \not\models_{\mathbb{T}} \varphi_{\text{mix}}$.

We close this section by defining the function `flatten` that rewrites hypertrace formulas into an equisatisfiable formula, exploiting the independence between variable valuations in traces assigned to unconstrained trace variables. For each trace variable in a set $\mathcal{V}$ and propositional variable, π and $x \in \mathcal{X}$, `flatten` introduces a new trace variable, π_x, which is used exclusively in predicates involving the corresponding variables (e.g., $x(\pi, i)$). For a given set of trace variables $\mathcal{V}$ and propositional variables $\mathcal{X}$, we define $\mathcal{V}_{\mathcal{X}} = \{\pi_x \mid \pi \in \mathcal{V} \text{ and } x \in \mathcal{X}\}$.

$$\begin{aligned}
\texttt{flatten}(\exists \pi \ \varphi, \{x_0, \ldots, x_n\}, \mathcal{V}^c) &= \exists \pi_{x_0} \ldots \exists \pi_{x_n} \texttt{flatten}(\varphi, \{x_0, \ldots, x_n\}, \mathcal{V}^c \cup \{\pi\}) \\
\texttt{flatten}(\exists \pi :: T \ \varphi, \mathcal{X}, \mathcal{V}^c) &= \exists \pi :: T \ \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c) \\
\texttt{flatten}(\exists i \ \varphi, \mathcal{X}, \mathcal{V}^c) &= \exists i \ \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c) \\
\texttt{flatten}(x(\pi, i), \mathcal{X}, \mathcal{V}^c) &= \begin{cases} x(\pi_x, i) & \text{if } \pi \in \mathcal{V}^c \\ x(\pi, i) & \text{otherwise} \end{cases} \\
\texttt{flatten}(\neg \varphi, \mathcal{X}, \mathcal{V}^c) &= \neg \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c) \\
\texttt{flatten}(\varphi \vee \varphi', \mathcal{X}, \mathcal{V}^c) &= \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c) \vee \texttt{flatten}(\varphi', \mathcal{X}, \mathcal{V}^c)
\end{aligned} \tag{6}$$

We prove below that `flatten` returns an equisatisfiable hypertrace formula.

▶ **Lemma 5.** *Let φ be a hypertrace formula over the set of propositional variables $\mathcal{X}$ and T be a set of traces over the same variables. Let $\Pi_{\mathbb{N}}$ and $\Pi_{\mathbb{T}}$ be trace and time assignments over the set of free variables in φ. Let $\mathcal{V}^c \subseteq free(\varphi)$ be a set of trace variables that are free in φ. For all trace assignments $\Pi'_{\mathbb{T}}$ over trace variables in $\mathcal{V}^c_{\mathcal{X}} \cup free(\varphi)$ agreeing with $\Pi_{\mathbb{T}}$ in its assignments of propositional variables x for the trace assigned to π (i.e., for all $\pi \in \mathcal{V}^c$ and $x \in \mathcal{X}$, $\Pi_{\mathbb{T}}(\pi) =_{\{x\}} \Pi'_{\mathbb{T}}(\pi_x)$) and otherwise having the same trace assignments of $\Pi_{\mathbb{T}}$ (i.e., for all $\pi \notin \mathcal{V}^c$, $\Pi_{\mathbb{T}}(\pi) = \Pi'_{\mathbb{T}}(\pi)$): $(\overline{T}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \varphi$ iff $(\overline{T}, (\Pi'_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c)$.*

[2] We slightly abuse notation by letting T denote both the set of traces generating the structure and the predicate determining which traces constrain the trace quantification. Note that, the set used to define the structure may be renamed, but the predicate T is true only for the traces in that set.

Proof. Let $\mathcal{X} = \{x_0, \ldots, x_n\}$. We prove the statement by structural induction on the formula.

Let us prove the $\Rightarrow$ direction. In the base case ($\varphi = x(\pi, i)$), $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} x(\pi, i)$ iff $(\Pi_{\mathbb{T}}(\pi), \Pi_{\mathbb{N}}(i)) \in X_x$. By definition of X_x (given by $\overline{\mathrm{T}}$), it holds that $(\Pi_{\mathbb{T}}(\pi), \Pi_{\mathbb{N}}(i)) \in X_x \Leftrightarrow \Pi_{\mathbb{T}}(\pi)[\Pi_{\mathbb{N}}(i)](x) = \mathit{true}$. If $\pi \in \mathcal{V}^c$, this is equivalent to $\Pi'_{\mathbb{T}}(\pi_x)[\Pi_{\mathbb{N}}(i)](x) = \mathit{true}$ (by definition of $\Pi'_{\mathbb{T}}$). Otherwise, it is equivalent to $\Pi'_{\mathbb{T}}(\pi)[\Pi_{\mathbb{N}}(i)](x) = \mathit{true}$ (also by definition of $\Pi'_{\mathbb{T}}$). Therefore, $(\overline{\mathrm{T}}, (\Pi'_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \texttt{flatten}(x(\pi, i), \mathcal{X}, \mathcal{V}^c)$. If φ is $i = j$ or $i < j$, the implication holds as these formulas depend only on $\Pi_{\mathbb{N}}$ which is not affected by changes from `flatten`.

In the induction step, cases for $\varphi_1 \vee \varphi_2$ and $\neg\varphi$ follow from the definition of $\models_{\mathbb{T}}$ and induction hypothesis. Assume the formula $\exists i\ \varphi$. Then, $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \exists i\ \varphi \Leftrightarrow \exists c \in \mathbb{N}$ s.t. $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}[i \mapsto c])) \models_{\mathbb{T}} \varphi$. From IH, $(\overline{\mathrm{T}}, (\Pi'_{\mathbb{T}}, \Pi_{\mathbb{N}}[i \mapsto c])) \models_{\mathbb{T}} \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c)$ which implies $(\overline{\mathrm{T}}, (\Pi'_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \texttt{flatten}(\exists i\ \varphi, \mathcal{X}, \mathcal{V}^c)$. Analogously for $\exists \pi{::}T$.

Finally, assume formula $\exists \pi\ \varphi$ and a set of trace variables $\mathcal{V}^c \subseteq \mathrm{free}(\exists \pi\ \varphi)$. This formula is satisfied iff there exists t s.t. $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}[\pi \mapsto t], \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \varphi$. We observe that $\pi \in \mathrm{free}(\varphi)$, as we assume there are no double binding of variables. Then, from IH, for $\mathcal{V}^{c\prime} = \mathcal{V}^c \cup \{\pi\}$, it follows that $(\overline{\mathrm{T}}, (\widetilde{\Pi_{\mathbb{T}}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^{c\prime})$ where $\widetilde{\Pi_{\mathbb{T}}}$ is such that for all $\pi' \in \mathcal{V}^c \cup \{\pi\}$ and $x \in \mathcal{X}$, $\Pi_{\mathbb{T}}[\pi \mapsto t](\pi') =_{\{x\}} \widetilde{\Pi_{\mathbb{T}}}(\pi_x)$ and otherwise the trace assignments are the same as $\Pi_{\mathbb{T}}$. Observe that $(\overline{\mathrm{T}}, (\Pi'_{\mathbb{T}}[\pi_{x_0} \mapsto \widetilde{\Pi_{\mathbb{T}}}(\pi_{x_0}), \ldots, \pi_{x_n} \mapsto \widetilde{\Pi_{\mathbb{T}}}(\pi_{x_n})], \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V} \cup \{\pi\})$. From the definition of $\models_{\mathbb{T}}$: $(\overline{\mathrm{T}}, (\Pi'_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \exists \pi_{x_0} \ldots \exists \pi_{x_n} \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}^c)$.

The proof for the $\Leftarrow$ direction is analogous to the $\Rightarrow$ direction, because the relation between $\Pi_{\mathbb{T}}$ and $\Pi'_{\mathbb{T}}$ is equational. ◀

It follows from the above lemma that all hypertrace formulas are equisatisfiable to their rewrite with `flatten`.

▶ **Corollary 6.** *Let φ be a closed hypertrace formula over the set of propositional variables $\mathcal{X}$: $[\![\varphi]\!] \neq \emptyset$ iff $[\![\texttt{flatten}(\varphi, \mathcal{X}, \emptyset)]\!] \neq \emptyset$.*

3.2 Satisfiability of Hypertrace Formulas

We are interested in the decidability of the satisfiability problem of hypertrace logic.

Satisfiability of Hypertrace Formulas

Let φ be a hypertrace formula over trace variables $\mathcal{V}_{\mathbb{T}}$, time variables $\mathcal{V}_{\mathbb{N}}$ and binary predicates defined from $\mathcal{X}$ (i.e., from the set $\{X_a \mid a \in \mathcal{X}\}$).
Is there a set of traces $\mathrm{T} \subseteq (\mathbf{2}^{\mathcal{X}})^\omega$ that is a model of φ; that is, $[\![\varphi]\!] \neq \emptyset$?

We present in Table 1 a summary of decidability results proved in this work for the satisfiability problem of hypertrace logic. We describe fragments by specifying patterns on its quantifiers: we denote $\mathbb{Q} \in \{\forall, \exists\}$, while $\mathbb{Q}_{\mathbb{N}_<}$ denotes time quantifiers, and $\mathbb{Q}_{\mathbb{T}}$ and $\mathbb{Q}_{\mathbb{T}}{::}T$ denotes unconstrained and constrained quantifiers, respectively. We then combine these symbols to define patterns as regular expressions.

4 Unconstrained Hypertrace Logic

We look into the fragment of hypertrace formulas without constrained quantifiers (i.e., no subformulas with shape $\mathbb{Q}\pi{:}T\ \varphi$), which we refer to as *unconstrained hypertrace logic*. We prove that unconstrained hypertrace logic is expressively equivalent to monadic second-order logic of order of one successor (S1S). The translation follows naturally from Lemma 5. Specifically,

Table 1 Summary of results on the decidability of hypertrace formulas satisfiability.

Trace-prefixed	
$\exists^*_{\mathbb{T}}(\exists_{\mathbb{T}}::T)^*(\forall_{\mathbb{T}}::T)^*\mathbb{Q}^*_{\mathbb{T}}\mathbb{Q}^*_{\mathbb{N}_<}$	Decidable [10] [Cor. 17]
$(\forall_{\mathbb{T}}::T)^2\exists_{\mathbb{T}}::T\ \mathbb{Q}^+_{\mathbb{N}_<}$	Undecidable [6] [Prop. 16]
Time-prefixed	
$\exists^*_{\mathbb{N}_<}\mathbb{E}^*_{\mathbb{T}}(\exists_{\mathbb{T}}::T)^*(\forall_{\mathbb{T}}::T)^*\mathbb{Q}^*_{\mathbb{T}}$	Decidable [Cor. 22]
$\exists_{\mathbb{N}_<}\forall_{\mathbb{N}_<}\exists^2_{\mathbb{N}_<}\forall_{\mathbb{N}_<}\forall_{\mathbb{T}}::T(\exists_{\mathbb{T}}::T)^2\exists_{\mathbb{T}}$	Undecidable [Thm. 23]

we show that quantification over variables in each unconstrained trace is equivalent to second-order quantification over the sets of time points at which the corresponding propositional variable holds in that trace.

4.1 Equivalence to S1S

To establish the equivalence between unconstrained hypertrace logic and `S1S`, we define a translation between unconstrained hypertrace formulas and models to their counterpart in `S1S`, and vice-versa. The translation from unconstrained hypertrace formulas to `S1S` rewrites the formulas using `flatten`, while reinterpreting each π_x as a second-order variable.

To translate from unconstrained formulas to `S1S` formulas, we define the substitution $\sigma = \{x(\pi_x, i) \mapsto \pi_x(i) \mid x \in \mathcal{X}, \pi \in \mathcal{V}_{\mathbb{T}}, i \in \mathcal{V}_{\mathbb{N}}\}$ and the rewriting function $\texttt{toS1S}(\varphi, \mathcal{X}) = \texttt{flatten}(\varphi, \mathcal{X}, \mathcal{V}_{\mathbb{T}} \cap \text{free}(\varphi))[\sigma]$ where φ is an unconstrained trace formula. This translation does not work for formulas with constrained trace quantifiers because, in general, we cannot assume T to be `S1S`-definable. The next step is to define a translation for trace assignments. A trace assignment $\Pi_{\mathbb{T}} : \mathcal{V}_{\mathbb{T}} \to (\mathbf{2}^{\mathcal{X}})^\omega$ is translated to an assignment over second-order variables where each variable is mapped to its support set. Formally, $\texttt{toSuppSet}(\Pi_{\mathbb{T}}) : \mathcal{V}_{\mathcal{X}} \to \mathbf{2}^{\mathbb{N}}$ where $\texttt{toSuppSet}(\Pi_{\mathbb{T}})(\pi_x) = \{i \mid x \in \Pi_{\mathbb{T}}(\pi)[i]\}$.

For the translation from `S1S` to unconstrained hypertrace formulas, each second-order variable X becomes the trace variable τ_X. Additionally, we use the standard translation of Succ using $\leq$.

$$
\begin{aligned}
\texttt{toHyper}(\exists X\ \varphi') &= \exists \pi_X\ \texttt{toHyper}(\varphi') & \qquad \texttt{toHyper}(X(i)) &= X(\pi_X, i) \\
\texttt{toHyper}(\exists i\ \varphi') &= \exists i\ \texttt{toHyper}(\varphi') & \texttt{toHyper}(i = j) &= (i = j) \\
\texttt{toHyper}(\neg\varphi') &= \neg\texttt{toHyper}(\varphi') & & \\
\texttt{toHyper}(\varphi_1 \vee \varphi_2) &= \texttt{toHyper}(\varphi_1) \vee \texttt{toHyper}(\varphi_2) & & \\
\texttt{toHyper}(\text{Succ}(i, i')) &= i < i' \wedge (\forall j\ (i < j \to i' \leq j)) & &
\end{aligned}
\tag{7}
$$

We translate second-order assignments, Π_2, to trace assignments as: $\texttt{toBool}(\Pi_2) : \mathcal{V}_{\mathbb{T}} \to (\mathbf{2}^{\mathcal{X}})^\omega$ where for all $i \in \mathbb{N}$ and $X \in \mathcal{X}$, $X \in (\texttt{toBool}(\Pi_2)(\pi_X))[i]$ iff $i \in \Pi_2(X)$.

▶ **Theorem 7.** *Let T be a set of traces over $\mathcal{X}$. For all unconstrained hypertrace formulas φ over $\mathcal{X}$: $(\overline{T}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \varphi$ iff $(\Pi_{\mathbb{N}}, \texttt{toSuppSet}(\Pi_{\mathbb{T}})) \models_{S1S} \texttt{toS1S}(\varphi, \mathcal{X})$. For all S1S formulas: $(\Pi_1, \Pi_2) \models_{S1S} \varphi$ iff $(\overline{T}, (\texttt{toBool}(\Pi_2), \Pi_1)) \models_{\mathbb{T}} \texttt{toHyper}(\varphi)$.*

Proof. We start by proving, by structural induction, the translation from unconstrained hypertrace formulas and trace assignments to `S1S` formulas and second-order assignments. For the base case, we need to prove $(\overline{\mathbb{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} x(\pi, i)$ iff $(\Pi_{\mathbb{N}}, \texttt{toSuppSet}(\Pi_{\mathbb{T}})) \models_{\texttt{S1S}} \pi_x(i)$. Given $k = \Pi_{\mathbb{N}}(i)$, we observe that, by definition, $x \in \Pi_{\mathbb{T}}(\pi)[k]$ iff $k \in (\texttt{toSuppSet}(\Pi_{\mathbb{T}}))(\pi_x)$. Hence, the base case holds. For the induction cases, we use the induction hypothesis and Lemma 5.

We consider now the translation from S1S formulas and second-order assignments to unconstrained hypertrace formulas and trace assignments. We have two base cases. We start by proving that $(\Pi_1, \Pi_2) \models_{\texttt{S1S}} X(i)$ iff $(\overline{\mathrm{T}}, (\texttt{toBool}(\Pi_2), \Pi_1) \models_{\mathbb{T}} X(\pi_X, i)$. This holds, because for $k = \Pi_{\mathbb{N}}(i)$, $X \in (\texttt{toBool}(\Pi_2)(\pi_X))[k]$ iff $k \in \Pi_2(X)$. The second base-case is $(\Pi_1, \Pi_2) \models_{\texttt{S1S}} \mathrm{Succ}(i,i')$ iff $(\overline{\mathrm{T}}, (\texttt{toBool}(\Pi_2), \Pi_1)) \models_{\mathbb{T}} i < i' \wedge (\forall j\ (i < j \rightarrow i' \leq j))$, holding by definition of $\leq$. For the induction cases, we use the induction hypothesis and Lemma 5. ◀

From the previous theorem and S1S having a decidable satisfiability checking problem [19], checking whether an unconstrained hypertrace formula is satisfiable is also decidable.

▶ **Corollary 8.** *The satisfiability problem for unconstrained hypertrace logic is decidable.*

4.2 Relation to Full Hypertrace Logic

In this section, we show that any hypertrace formula with a single alternation from an existential to a universal constrained quantifier can be rewritten into an equisatisfiable formula without constrained quantifiers. We prove our results by adapting techniques introduced in [6, 5] to rewrite HyperQPTL formulas into QPTL. We start by showing how to eliminate the universal constrained trace quantifiers.

▶ **Lemma 9.** *Let* $\varphi = \overrightarrow{\mathbb{E}}\ \exists \pi_1 ::T \ldots \exists \pi_n ::T\ \forall \pi'_1 ::T \ldots \forall \pi'_m ::T\ \overrightarrow{\mathbb{Q}}\ \varphi_{qf}$ *be a hypertrace formula in prenex normal form s.t.* $\mathbb{E}$ *is any combination of existential time or unconstrained trace quantifiers,* $\overrightarrow{\mathbb{Q}}$ *is any combination of time or unconstrained trace quantifiers and* φ_{qf} *is a quantifier-free hypertrace formula. Let*

$$\texttt{removeForAll}(\varphi) = \overrightarrow{\mathbb{E}}\ \exists \pi_1 ::T \ldots \exists \pi_n ::T \bigwedge_{j_1=1}^{n} \ldots \bigwedge_{j_m=1}^{n} \overrightarrow{\mathbb{Q}}\ \varphi_{qf}[\pi'_1 \mapsto \pi_{j_1}, \ldots, \pi'_m \mapsto \pi_{j_m}]$$

where $\varphi_{qf}[\pi'_1 \mapsto \pi_{j_1}, \ldots, \pi'_m \mapsto \pi_{j_m}]$ *is the formula obtained by substituting* π'_i *with* π_{j_i}*, for all* $1 \leq i \leq m$*, in* φ_{qf}*. Then,* $[\![\varphi]\!] \neq \emptyset$ *iff* $[\![\texttt{removeForAll}(\varphi)]\!] \neq \emptyset$*.*

Proof. A set of trace variables $\mathcal{V}$ and a trace assignment $\Pi_{\mathbb{T}}$ that includes assignments for all variables in $\mathcal{V}$ (i.e., $\mathcal{V} \subseteq \mathrm{Dom}(\Pi_{\mathbb{T}})$) define the set of traces $\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V})} = \{\Pi_{\mathbb{T}}(\pi) \mid \pi \in \mathcal{V}\}$. For a set of free variables and assignment, $\mathcal{V}$ and $\Pi_{\mathbb{T}}$, when we evaluate the subformula starting with the universal constrained quantifier against the set of traces $\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V})}$, we can remove the universal quantifiers by considering all possible combinations of assignments to the traces in $\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V})}$. Formally, for any hypertrace formula $\forall \pi'_1 ::T \ldots \forall \pi'_m ::T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}}$ where $\overrightarrow{\mathbb{Q}}$ is any combination of time or unconstrained trace quantifiers, φ_{qf} is a quantifier-free hypertrace formula, and $\mathcal{V} = \{\pi_1, \ldots, \pi_n\}$ s.t. $\mathcal{V} \subseteq \mathrm{free}(\forall \pi'_1 ::T \ldots \forall \pi'_m ::T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}})$, the equivalence below follows from definitions:

$$(\overline{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V})}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \forall \pi'_1 ::T \ldots \forall \pi'_m ::T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}} \text{ iff} \tag{8}$$
$$(\overline{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V})}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \bigwedge_{j_1=1}^{n} \ldots \bigwedge_{j_m=1}^{n} \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}}[\pi'_1 \mapsto \pi_{j_1}, \ldots, \pi'_m \mapsto \pi_{j_m}].$$

Before we prove the main claim, we need to prove that the set of models of a hypertrace formulas without existential constrained quantifiers is subset-closed.

▷ Claim 1. Let φ be a hypertrace formula in prenex and negation normal form (i.e., all quantifiers at the beginning of the formula and negation only at the atomic level) without existential constrained trace quantifiers. For all sets of traces T and its subsets $\mathrm{T}' \subseteq \mathrm{T}$, and assignments $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}}$: if $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \varphi$, then $(\overline{\mathrm{T}'}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \varphi$.

Proof of Claim 1. We prove by structural induction on φ. For the bases cases, we observe that $\models_{\mathbb{T}}$ is independent of the set of traces, hence, as the assignments are preserved, the property holds. The induction cases $\varphi \vee \varphi'$, $\varphi \wedge \varphi'$, $\exists i\ \varphi$ and $\forall i\ \varphi$ follow directly from definitions and induction hypothesis. The only case remaining is $\forall \pi{::}T\ \varphi$. Consider arbitrary set of traces T and $\mathrm{T}' \subseteq \mathrm{T}$ and assignments $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}}$. We assume that $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \forall \pi{::}T\ \varphi$, which, by definition of $\models_{\mathbb{T}}$, is equivalent to: for all traces $\tau \in \mathrm{T}$, $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}[\pi \mapsto \tau], \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \varphi$. By induction hypothesis and $\mathrm{T}' \subseteq \mathrm{T}$, for all traces $\tau \in \mathrm{T}'$, $(\overline{\mathrm{T}'}, (\Pi_{\mathbb{T}}[\pi \mapsto \tau], \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \varphi$. Hence, by definition of $\models_{\mathbb{T}}$, $(\overline{\mathrm{T}'}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}) \models_{\mathbb{T}} \forall \pi{::}T\ \varphi$. ◁

We prove now each side of the implication. We assume without loss of generality that the formula is in prenex and negation normal form. In what follows, $\mathcal{V}_\exists = \{\pi_1, \ldots, \pi_n\}$. We start by proving that $[\![\varphi]\!] \neq \emptyset \Rightarrow [\![\texttt{removeForAll}(\varphi)]\!] \neq \emptyset$:

$[\![\varphi]\!] \neq \emptyset \Leftrightarrow \text{exists } \mathrm{T} \in [\![\overrightarrow{\mathbb{E}}\ \exists \pi_1{::}T \ldots \exists \pi_n{::}T\ \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}}]\!] \overset{\text{def.} \models_{\mathbb{T}}}{\Leftrightarrow}$

exists $\mathrm{T}, \Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}}$ s.t. $\Pi_{\mathbb{T}}(\pi) \in \mathrm{T}$ with $\pi \in \mathcal{V}_\exists$:

$(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}}\ \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}} \overset{\mathrm{T} \subseteq \mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)}, \text{Claim 1}}{\Rightarrow}$

exists $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}} : (\overline{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}} \overset{(8)}{\Leftrightarrow}$

exists $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}}$:

$$(\overline{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \bigwedge_{j_1=1}^{n} \ldots \bigwedge_{j_m=1}^{n} \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}}[\pi'_1 \mapsto \pi_{j_1}, \ldots, \pi'_m \mapsto \pi_{j_m}] \Leftrightarrow$$

exists $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}} : \mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)} \models_{\mathbb{T}} \texttt{removeForAll}(\varphi) \quad \Rightarrow \quad [\![\texttt{removeForAll}(\varphi)]\!] \neq \emptyset$

And now, we prove $[\![\texttt{removeForAll}(\varphi)]\!] \neq \emptyset \Rightarrow [\![\varphi]\!] \neq \emptyset$:

$[\![\texttt{removeForAll}(\varphi)]\!] \neq \emptyset \Leftrightarrow$

$$\text{exists } \mathrm{T} \in [\![\overrightarrow{\mathbb{E}}\ \exists \pi_1{::}T \ldots \exists \pi_n{::}T \bigwedge_{j_1=1}^{n} \ldots \bigwedge_{j_m=1}^{n} \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}}[\pi'_1 \mapsto \pi_{j_1}, \ldots, \pi'_m \mapsto \pi_{j_m}]]\!] \overset{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)} \subseteq \mathrm{T}}{\underset{\text{Claim 1, (8)}}{\Rightarrow}}$$

exists $\Pi_{\mathbb{T}}$ and $\Pi_{\mathbb{N}} : (\overline{\mathrm{T}_{(\Pi_{\mathbb{T}}, \mathcal{V}_\exists)}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}})) \models_{\mathbb{T}} \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{\mathrm{qf}} \Rightarrow \text{exists } \mathrm{T} \in [\![\varphi]\!]$ ◀

▶ **Example 11.** for the formula: $\varphi = \exists \pi_1 \exists \pi_2 \forall \pi_3 \exists i \exists j\ a(\pi_1, i) \wedge \neg a(\pi_2, i) \wedge b(\pi_3, j)$, we have

$$\texttt{removeForAll}(\varphi) = \exists \pi_1 \exists \pi_2 \\ (\exists i \exists j\ a(\pi_1, i) \wedge \neg a(\pi_2, i) \wedge b(\pi_1, j)) \wedge (\exists i \exists j\ a(\pi_1, i) \wedge \neg a(\pi_2, i) \wedge b(\pi_2, j))$$

which can be simplified to $\exists \pi_1 \exists \pi_2 \exists i\ a(\pi_1, i) \wedge \neg a(\pi_2, i) \wedge (\exists j\ b(\pi_1, j)) \wedge (\exists j\ b(\pi_2, j))$.

We observe that hypertrace formulas with only existential constrained trace quantifiers are equisatisfiable to an unconstrained formula. Intuitively, each existential trace quantifier can be instantiated independently of the others.

▶ **Lemma 12.** *Let $\varphi = \overrightarrow{\mathbb{E}}\ \exists \pi_1{::}T \ldots \exists \pi_n{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{qf}$ be a hypertrace formula in prenex normal form s.t. $\overrightarrow{\mathbb{E}}$ is a any combination of existential time or unconstrained trace quantifiers, $\overrightarrow{\mathbb{Q}}$ is any combination of time or unconstrained trace quantifiers and φ_{qf} is a quantifier-free hypertrace formula. $[\![\varphi]\!] \neq \emptyset$ iff $[\![\overrightarrow{\mathbb{E}}\ \exists \pi_1 \ldots \exists \pi_n\ \overrightarrow{\mathbb{Q}}\ \varphi_{qf}]\!] \neq \emptyset$.*

Finally, for all hypertrace formulas where all universal constrained trace quantifiers are only followed by existential constrained quantifiers, we can apply the rewrite described in Lemma 9 and Lemma 12 to get an equisatisfiable unconstrained hypertrace formula.

▶ **Theorem 13.** *Let* $\varphi = \overrightarrow{\mathbb{E}}\ \exists \pi_1{::}T \ldots \exists \pi_n{::}T\ \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}}\ \varphi_{qf}$ *be a hypertrace formula in prenex normal form s.t.* $\mathbb{E}$ *is any combination of existential time or unconstrained trace quantifiers,* $\overrightarrow{\mathbb{Q}}$ *is any combination of time or unconstrained trace quantifiers and* φ_{qf} *is a quantifier-free hypertrace formula. There exists an unconstrained hypertrace formula* φ_u *s.t.:* $[\![\varphi]\!] \neq \emptyset$ *iff* $[\![\varphi_u]\!] \neq \emptyset$.

5 Trace-prefixed Hypertrace Logic

In this section, we consider the fragment of FO$[<, \mathbb{T}]$ where trace quantifiers come before time quantifiers. We call this the *trace-prefixed hypertrace logic*, denoted $\mathbf{T}$-FO$[<, \mathbb{T}]$. Formally, trace-prefixed formulas $\varphi \in \mathbf{T}$-FO$[<, \mathbb{T}]$ are defined by the grammar:

$$\varphi ::= \exists \pi\, \varphi \mid \exists \pi{::}T\, \varphi \mid \neg\varphi \mid \psi \qquad \psi ::= \exists i\ \psi \mid \psi \vee \psi \mid \neg\psi \mid i < i \mid i = i \mid X(\pi, i)$$

where π is a trace variable, i is a time variable and X is a binary predicate.

We observe that $\mathbf{T}$-FO$[<, \mathbb{T}]$ is orthogonal to unconstrained hypertrace logic. While the unconstrained fragment allows any mix of (unconstrained) trace and time quantifiers, $\mathbf{T}$-FO$[<, \mathbb{T}]$ requires trace quantifiers to come first. On the other hand, $\mathbf{T}$-FO$[<, \mathbb{T}]$ introduces constrained trace quantifiers, which the unconstrained version does not support.

▶ **Example 14.** We can express *bounded promptness* in $\mathbf{T}$-FO$[<, \mathbb{T}]$:

$$\exists \pi\ \forall \pi'{::}T\ \exists i\ \forall j\ \big((j < i \rightarrow \neg p(\pi, j)) \wedge p(\pi, i) \wedge q(\pi', i)\big). \tag{9}$$

In this property, the unconstrained trace is used to guess a synchronization point (i.e., the time when p becomes true in π) where all traces agree with the value of proposition q.

In [1], the authors study the trace-prefixed hypertrace logic for the fragment of FO$[<, \mathbb{T}]$ with only constrained trace quantifiers, which we refer to as $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$. They prove that $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ is expressively equivalent to HyperLTL. From this result, we can prove that adding unconstrained quantifiers to $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ extends its expressive power.

▶ **Proposition 15.** $\boldsymbol{T}$*-FO*$[<, \mathbb{T}]$ *is strictly more expressive than* $\boldsymbol{T}$*-FO*$[<, \mathbb{T}{::}T]$.

Proof. Clearly, $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ is a fragment of $\mathbf{T}$-FO$[<, \mathbb{T}]$. To prove that $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ is not equally expressive to $\mathbf{T}$-FO$[<, \mathbb{T}]$, we observe that bounded promptness (9) is not expressible in HyperLTL [4] and, from $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ being equally expressive to HyperLTL [1], it is also not expressible in $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$. ◀

We can also use $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ to get our first undecidability result for $\mathbf{T}$-FO$[<, \mathbb{T}]$.

▶ **Proposition 16.** *Let* $\varphi = \forall \pi_1{::}T\ \forall \pi_2{::}T\ \exists \pi'_1{::}T\ \psi$ *where* ψ *has only time quantifiers. It is undecidable to check whether* $[\![\varphi]\!] \neq \emptyset$.

Proof. We prove by reduction of the HyperLTL satisfiability problem for formulas with quantifier pattern $\forall\forall\exists$ to the problem of checking for satisfiability of hypertrace formulas with quantifier pattern $\forall{::}T\ \forall{::}T\ \exists{::}T$. Consider an arbitrary HyperLTL formula $\varphi = \forall \pi_1 \forall \pi_2 \exists \pi'_1 \psi$ where ψ has only temporal operators. By $\mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ being expressively equivalent to HyperLTL, there exists $\varphi' \in \mathbf{T}$-FO$[<, \mathbb{T}{::}T]$ s.t. $[\![\varphi]\!] = [\![\varphi']\!]$. Using the computable translation in [1], $\varphi' = \forall \pi_1{::}T\ \forall \pi_2{::}T\ \exists \pi'_1{::}T \psi'$ where ψ' has only temporal quantifiers. From [6, 14], it is undecidable to check for satisfiability of HyperLTL formulas with quantifier pattern $\forall\forall\exists$. Hence, it is also not possible to decide $[\![\varphi]\!] \neq \emptyset$. ◀

5.1 Satisfiability

From Theorem 13 and Corollary 8, we get our first fragment of $\mathbf{T}$-FO$[<,\mathbb{T}]$ with a decidable satisfiability problem: the fragment where constrained trace quantifiers is of shape $\exists^*\forall^*$. This matches the known decidable fragment for HyperQPTL [10].

▶ **Corollary 17.** *Let $\varphi = \overrightarrow{\mathbb{E}_\mathbb{T}}\ \exists \pi_1{::}T \ldots \exists \pi_n{::}T\ \forall \pi'_1{::}T \ldots \forall \pi'_m{::}T\ \overrightarrow{\mathbb{Q}_\mathbb{T}}\ \overrightarrow{\mathbb{Q}_{\mathbb{N}_<}}\ \varphi_{qf}$ be a hypertrace formula in prenex normal form s.t. $\overrightarrow{\mathbb{E}_\mathbb{T}}$ is a sequence of existential unconstrained trace quantifiers, $\overrightarrow{\mathbb{Q}_\mathbb{T}}$ and $\overrightarrow{\mathbb{Q}_{\mathbb{N}_<}}$ are any combination of time and unconstrained trace quantifiers, respectively, and φ_{qf} is a quantifier-free. It is decidable to check whether $[\![\varphi]\!] \neq \emptyset$.*

Unconstrained trace quantifiers not only extend the expressivity of $\mathbf{T}$-FO$[<,\mathbb{T}]$ compared to $\mathbf{T}$-FO$[<,\mathbb{T}{::}T]$ but can also be used to seamlessly define a semi-decision procedure to check for the unsatisfiability of $\mathbf{T}$-FO$[<,\mathbb{T}{::}T]$ formulas. In the proposition below, we prove that removing constraints from existentially quantified trace variables preserves the models of the constrained formula. We can use the contrapositive of this proposition to determine the unsatisfiability of constrained hypertrace formulas.

▶ **Proposition 18.** *Let $\varphi = \forall \pi_0{::}T\ \exists \pi'_0{::}T \ldots \forall \pi_k{::}T\ \exists \pi'_k{::}T\ \psi$ be a hypertrace formula where ψ is quantifier-free, and T be a set of traces. If $T \models_\mathbb{T} \varphi$, then $T \models_\mathbb{T} \forall \pi_0{::}T\ \exists \pi'_0 \ldots \forall \pi_k{::}T\ \exists \pi'_k\ \psi$.*

Proof. The statement follows directly from T being a subset of all possible traces and, thus, any assignment over T is also an assignment over the set of all traces. ◀

5.2 Equivalence to HyperQPTL

We prove that, for sets of infinite traces, trace-prefixed hypertrace logic is expressively equivalent to HyperQPTL [17]. HyperQPTL formulas φ are defined by the grammar:

$$\psi ::= \mathbf{X}\psi \mid \psi\ \mathbf{U}\ \psi \mid \neg\psi \mid \psi \vee \psi \mid q \mid a_\pi \qquad \varphi ::= \exists q\, \varphi \mid \forall q\, \varphi \mid \exists \pi\, \varphi \mid \forall \pi\, \varphi \mid \psi$$

where $\pi \in \mathcal{V}_\mathbb{T}$ is a trace variable and $q, a \in \mathcal{X}$ are propositional variables, where $\mathcal{V}_\mathbb{T} \cap \mathcal{X} = \emptyset$. We evaluate HyperQPTL formulas over a set of traces and a trace assignment $\Pi : \mathcal{V}_\mathbb{T} \to (\mathbf{2}^\mathcal{X})^\omega$, relative to a time point $i \in \mathbb{N}$:

$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \exists q\ \psi$ iff there exists $\tau \in (\mathbf{2}^{\{q\}})^\omega : (\Pi[\pi_q \mapsto \tau], \mathrm{T}, i) \models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \forall q\ \psi$ iff for all $\tau \in (\mathbf{2}^{\{q\}})^\omega : (\Pi[\pi_q \mapsto \tau], \mathrm{T}, i) \models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \exists \pi\ \psi$ iff there exists $\tau \in \mathrm{T} : (\Pi[\pi \mapsto \tau], \mathrm{T}, i) \models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \forall \pi\ \psi$ iff for all $\tau \in \mathrm{T} : (\Pi[\pi \mapsto \tau], \mathrm{T}, i) \models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} q$ iff $q \in \Pi(\pi_q)[i]$; $(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} a_\pi$ iff $a \in \Pi(\pi)[i]$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \neg\psi$ iff $(\Pi, \mathrm{T}, i) \not\models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \psi_1 \vee \psi_2$ iff $(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \psi_1$ or $(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \psi_2$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \mathbf{X}\psi$ iff $(\Pi, \mathrm{T}, i+1) \models_{\mathtt{HQ}} \psi$;
$(\Pi, \mathrm{T}, i) \models_{\mathtt{HQ}} \psi_1\ \mathbf{U}\ \psi_2$ iff exists $j \geq i : (\Pi, \mathrm{T}, j) \models_{\mathtt{HQ}} \psi_2$ and all $i \leq j' < j$ $(\Pi, \mathrm{T}, j') \models_{\mathtt{HQ}} \psi_1$.

A set T of traces is a model of a HyperQPTL formula φ, denoted $\mathrm{T} \models_{\mathtt{HQ}} \varphi$, iff there exists an assignment Π such that $(\Pi, \mathrm{T}, 0) \models_{\mathtt{HQ}} \varphi$. For all closed formulas φ, $\mathrm{T} \models_{\mathtt{HQ}} \varphi$ iff $(\Pi^\emptyset, \mathrm{T}, 0) \models_{\mathtt{HQ}} \varphi$, where $\Pi^\emptyset$ is the empty assignment.

▶ Remark 19. In this work, we interpret HyperQPTL formulas as introduced in [17, 5]. An alternative definition found in the literature uniformly instantiates quantified propositions across the set of traces used as the model [10, 7, 20, 18]. The semantics we adopt in this work subsumes the uniform interpretation; that is, we can simulate the uniform interpretation by rewriting the HyperQPTL formula interpreted under the uniform semantics.

For a set of traces T and an assignment Π for variables in $\mathcal{V}$, we define its *flattening* to a trace as: $a_\pi \in \langle\Pi\rangle[i]$ iff $a \in \Pi(\pi)[i]$ and $q \in \langle\Pi\rangle[i]$ iff $q \in \Pi(\pi_q)[i]$ for all $i \in \mathbb{N}$, trace variables $\pi \in \mathcal{V}$ and propositional variables $a, q \in \mathcal{X}$. A trace assignment satisfies a quantifier-free HyperQPTL formula iff its flattening satisfies the same formula under the LTL semantics.

▶ **Proposition 20.** *Let φ a quantifier-free* HyperQPTL *formula. For all $i \in \mathbb{N}$, all trace sets T, and all trace assignments* Π, $(\Pi, T, i) \models_{\texttt{HQ}} \varphi$ *iff* $\langle\Pi\rangle[i \ldots] \models_{\texttt{LTL}} \varphi$.

▶ **Theorem 21.** *For all* HyperQPTL *sentences φ_H there exists a trace-prefixed hypertrace sentence φ such that for all sets of infinite traces $T \subseteq (\mathbf{2}^{\mathcal{X}})^\omega$, $T \models_{\texttt{HQ}} \varphi_H$ iff $T \models_{\mathbb{T}} \varphi$. For all trace-prefixed hypertrace sentences φ there exists a* HyperQPTL *sentence φ_H such that for all sets of infinite traces $T \subseteq (\mathbf{2}^{\mathcal{X}})^\omega$, $T \models_{\texttt{HQ}} \varphi_H$ iff $T \models_{\mathbb{T}} \varphi$.*

Proof. We denote by $\texttt{LTLtoFO}(\psi)$ and $\texttt{FOtoLTL}(\psi)$ the translation from LTL formulas to FO[<] and vice-versa given by LTL and FO[<] being equivalent [9].

We start with the translation from HyperQPTL formulas to an equivalent $\mathbf{T}$-FO[$<, \mathbb{T}$] formula. We first define the translation of $\varphi \in$ HyperQPTL for its different quantifiers:

$$\texttt{tr}_H^{\text{quant}}(\varphi) = \begin{cases} \varphi & \text{if } \varphi \text{ is quantifier-free} \\ \mathbb{Q}\pi{::}T\ \texttt{tr}_H^{\text{quant}}(\varphi') & \text{if } \varphi = \mathbb{Q}\pi\ \varphi', \mathbb{Q} \in \{\forall, \exists\} \text{ and } \pi \in \mathcal{V}_{\mathbb{T}} \\ \mathbb{Q}\pi_q\ \texttt{tr}_H^{\text{quant}}(\varphi') & \text{if } \text{ if } \varphi = \mathbb{Q}q\ \varphi', \mathbb{Q} \in \{\forall, \exists\} \text{ and } q \in \mathcal{X} \end{cases} \tag{10}$$

With $\texttt{LTLtoFO}(\psi)$ all propositional variables a_π in ψ are mapped to $P_{a_\pi}(i)$, while all q are mapped to $P_q(i)$. We define the substitution from these predicates to the binary predicates of FO[$<, \mathbb{T}$]: $\sigma = \{P_{a_\pi}(i) \mapsto X_a(\pi, i) \mid a \in \mathcal{X}, \pi \in \mathcal{V}_{\mathbb{T}}, i \in \mathcal{V}_{\mathbb{N}}\} \cup \{P_q(i) \mapsto X_q(\pi_q, i) \mid q \in \mathcal{X}, \pi \in \mathcal{V}_{\mathbb{T}}, i \in \mathcal{V}_{\mathbb{N}}\}$. The translation from HyperQPTL formulas to $\mathbf{T}$-FO[$<, \mathbb{T}$] is defined below, where $\psi \in$ LTL, $x_i \in \mathcal{V}_{\mathbb{T}} \cup \mathcal{X}$, for $1 \leq i \leq n$, and $\mathbb{Q} \in \{\forall, \exists\}$:

$$\texttt{tr}_H(\mathbb{Q}x_1 \ldots \mathbb{Q}x_n\ \psi) = \texttt{tr}_H^{\text{quant}}(\mathbb{Q}x_1 \ldots \mathbb{Q}x_n\ ((\texttt{LTLtoFO}(\psi))[\sigma])). \tag{11}$$

Given a trace assignment, Π, and $k \in \mathbb{N}$ we denote Π_k any assignment satisfying $q \in \Pi_k(\pi_q)[i]$ iff $q \in \Pi(\pi_q)[i + k]$, and otherwise, $\Pi_k(\pi) = (\Pi(\pi))[k \ldots]$. We prove in the extended version, using the equivalence between LTL and FO[<] [9] and Prop. 20, that for all HyperQPTL formula $\varphi = \mathbb{Q}x_1 \ldots \mathbb{Q}x_n\ \psi$ where $x_i \in \mathcal{V}_{\mathbb{T}} \cup \mathcal{X}$, for $1 \leq i \leq n$, and $\mathbb{Q} \in \{\forall, \exists\}$: $(\Pi, \mathrm{T}, k) \models_{\texttt{HQ}} \varphi$ iff $(\overline{\mathrm{T}[k \ldots]}, (\Pi_k, \Pi_{\mathbb{N}}^{\emptyset})) \models_{\mathbb{T}} \texttt{tr}_H(\varphi)$, where $\Pi_{\mathbb{N}}^{\emptyset}$ is the empty time assignment.

For the translation from trace-prefixed hypertrace formulas to HyperQPTL, we use Lemma 5, and give a translation from flattened hypertrace formulas to HyperQPTL. As before we define a substitution $\sigma'' = \{X_a(\pi, i) \mapsto P_{a_\pi}(i) \mid a \in \mathcal{X}, \pi \in \mathcal{V}_{\mathbb{T}}, i \in \mathcal{V}_{\mathbb{N}}\} \cup \{X_q(\pi_q, i) \mapsto P_q(i) \mid q \in \mathcal{X}, \pi \in \mathcal{V}_{\mathbb{T}}, i \in \mathcal{V}_{\mathbb{N}}\}$. The translation is defined as: $\texttt{tr}_{\mathbb{T}}(\varphi) = \overrightarrow{\mathbb{Q}}(\texttt{FOtoLTL}(\psi'[\sigma'']))$ with $\overrightarrow{\mathbb{Q}}\psi' = \texttt{flatten}(\varphi, \mathcal{X}, \emptyset)$ where $\overrightarrow{\mathbb{Q}}$ is a sequence of constrained and unconstrained trace quantifiers and ψ' is a hypertrace formula with no trace quantifiers.

We prove that for all hypertrace formulas φ with only free trace variables, for all sets of traces T and assignments $\Pi_{\mathbb{T}}$: $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}^{\emptyset})) \models_{\mathbb{T}} \varphi$ iff $(\Pi'_{\mathbb{T}}, \mathrm{T}, 0) \models_{\texttt{HQ}} \texttt{tr}_{\mathbb{T}}(\varphi)$, where $\Pi'_{\mathbb{T}}(\pi_q) = (\Pi_{\mathbb{T}}(\pi_q)[0] \cap \{q\})(\Pi_{\mathbb{T}}(\pi_q)[1] \cap \{q\}) \ldots$. For the base case, where ψ' has no trace quantifiers and all time quantifiers are bounded: $(\overline{\mathrm{T}}, (\Pi_{\mathbb{T}}, \Pi_{\mathbb{N}}^{\emptyset})) \models_{\mathbb{T}} \psi'$ iff $(\overline{\langle\Pi_{\mathbb{T}}\rangle}, \Pi_{\mathbb{N}}^{\emptyset}) \models$

$\psi'[\sigma']$, follows from an analogous proof from the previous case. As the translation from $\Pi_{\mathbb{T}}$ to $\Pi'_{\mathbb{T}}$ does not change the valuation of q in the trace assigned to π_q and by [9], $(\overline{\langle \Pi_{\mathbb{T}} \rangle}, \Pi^{\emptyset}_{\mathbb{N}}) \models \psi'[\sigma']$ iff $\langle \Pi'_{\mathbb{T}} \rangle \models_{\texttt{LTL}} \texttt{FOtoLTL}(\psi'[\sigma'])$. By Proposition 20, it is equivalent to $(\Pi'_{\mathbb{T}}, \mathrm{T}, 0) \models_{\texttt{HQ}} \texttt{FOtoLTL}(\psi'[\sigma'])$. The induction cases (i.e., constrained and unconstrained quantifiers) follow from induction hypothesis and definitions. ◀

6 Time-prefixed Hypertrace Logic

We consider now time-prefixed hypertrace logic, with formulas defined by the grammar:

$$\varphi ::= \exists i\ \varphi \mid \neg\varphi \mid \psi \qquad \psi ::= \exists \pi\ \psi \mid \exists \pi :: T\ \psi \mid \psi \vee \psi \mid \neg\psi \mid i < i \mid i = i \mid X(\pi, i)$$

where π is a trace variable, i is a time variable and X is a binary predicate. To the best of our knowledge, there is no formalism in the literature allowing to specify hyperproperties by quantifying over time before traces, while supporting arbitrary time and trace quantifiers. The relative expressiveness between the trace-prefix and time-prefix fragments, however, remains an open problem.

As for the previously studied fragments, we use Theorem 13, and Corollary 8 to identify a fragment of the time-prefixed hypertrace logic with decidable satisfiability problem.

▶ **Corollary 22.** *Let* $\varphi = \overrightarrow{\mathbb{E}_{\mathbb{N}_<}}\ \exists \pi_1 :: T \ldots \exists \pi_n :: T\ \forall \pi'_1 :: T \ldots \forall \pi'_m :: T\ \overrightarrow{\mathbb{Q}_{\mathbb{T}}}\ \varphi_{qf}$ *be a hypertrace formula in prenex normal form s.t.* $\overrightarrow{\mathbb{E}_{\mathbb{N}_<}}$ *is a sequence of existential time quantifiers,* $\overrightarrow{\mathbb{Q}_{\mathbb{T}}}$ *is any combination of time and unconstrained trace quantifiers, respectively, and* φ_{qf} *is a quantifier-free. It is decidable to check whether* $[\![\varphi]\!] \neq \emptyset$.

In our theorem below, we prove that the satisfiability problem for arbitrary time-prefixed formulas is undecidable. We prove our result with a reduction from the (non)-halting problem for 2-counter Minsky machines.

A *2-counter Minsky machine* is defined by a tuple $\mathcal{M} = (Q, \Delta, \hat{q})$ where Q is a finite set of states with $\hat{q} \in Q$ being the initial state, and $\Delta \subseteq Q \times \{1, 2\} \times \{\texttt{inc}, \texttt{dec}, \texttt{isZero}\} \times Q$ being the transition relation. For all $n, n' \in \mathbb{N}$, we write: $n \xrightarrow{\texttt{isZero}} n'$ iff $n = n' = 0$; $n \xrightarrow{\texttt{inc}} n'$ iff $n' = n + 1$; and $n \xrightarrow{\texttt{dec}} n'$ iff $n' = n - 1$. We observe that `dec` is only allowed when $n > 0$. A configuration is a tuple with a state and the current value of the two counters. Given one of the counters $c \in \{1, 2\}$ we refer to the other counter as $\bar{c} = 3 - c$. We say that there is a transition between two configurations (q, n_1, n_2) and (q', n'_1, n'_2) iff there exists $(q, c, op, q') \in \Delta$ s.t. $n_c \xrightarrow{op} n'_c$ and, for the other counter, $n_{\bar{c}} = n'_{\bar{c}}$. A computation is a sequence of configurations connected by transitions. It is undecidable to check whether an arbitrary 2-counter Minsky machine has an infinite computation [15].

▶ **Theorem 23.** *Let* $\mathbb{Q} \in \exists_{\mathbb{N}_<} \forall_{\mathbb{N}_<} \exists^2_{\mathbb{N}_<} \forall_{\mathbb{N}_<} \forall_{\mathbb{T}} :: T (\exists_{\mathbb{T}} :: T)^2 \exists_{\mathbb{T}}$ *and* $\varphi = \mathbb{Q}\varphi'$ *be a time-prefixed hypertrace formula where* φ' *is quantifier-free. It is undecidable to check whether* $[\![\varphi]\!] \neq \emptyset$.

Proof. In our reduction from the (non)-halting problem for 2-counter Minsky machines, each trace encodes one configuration and the transition relation for the next step of the computation. Traces include the following propositional variables to capture configurations, for a given set of states Q: $\mathcal{X}_{\text{config}} = Q \cup \{\texttt{mem}_1, \texttt{mem}_2\}$ with $\texttt{mem}_1$ and $\texttt{mem}_2$ encoding the current value in their respective counters. We use $\mathcal{X}_{\text{transition}} = \{q_{\text{to}}, q_{\text{from}} \mid q \in Q\} \cup \{\texttt{inc}, \texttt{dec}, \texttt{isZero}, \texttt{to}_1, \texttt{to}_2\}$ to specify transitions where $\texttt{to}_1$ and $\texttt{to}_2$ represent the counters to be updated. The values in each counter are represented in unary and so incrementing the counter c amounts to making $\texttt{mem}_i$ true for one more step, while decrementing removes the last true valuation. We use an

unconstrained trace quantifier to guess the time point where this increment or decrement takes place. This guess is guided by two propositional variables: $\mathcal{X}_{\text{guess}} = \{\texttt{guess}, \texttt{guessed}\}$. Hence, our traces are defined over $\mathcal{X} = \mathcal{X}_{\text{config}} \cup \mathcal{X}_{\text{transition}} \cup \mathcal{X}_{\text{guess}}$.

We start by defining useful formulas to encode requirements of well-formed traces, where $\texttt{exactlyOne}(\mathcal{Y})$ is true when only one of the propositions in $\mathcal{Y}$ holds:

- At each time i the trace π defines an unique transition:

$$\texttt{singleTr}(Q,\pi,i) \stackrel{\text{def}}{=} \texttt{exactlyOne}(\{q_{\text{to}}(\pi,i) \mid q \in Q\}) \ \wedge \texttt{exactlyOne}(\{q_{\text{from}}(\pi,i) \mid q \in Q\}) \wedge \texttt{exactlyOne}(\{\texttt{to}_1(\pi,i), \texttt{to}_2(\pi,i)\}) \wedge \texttt{exactlyOne}(\{\texttt{inc}(\pi,i), \texttt{dec}(\pi,i), \texttt{isZero}(\pi,i)\}) \tag{12}$$

- for times i and j the trace π defines the same transition:

$$\texttt{sameTr}(Q,\pi,i,j) \stackrel{\text{def}}{=} \bigwedge_{q \in Q} (q_{\text{to}}(\pi,i) \leftrightarrow q_{\text{to}}(\pi,j)) \wedge \bigwedge_{q \in Q} (q_{\text{from}}(\pi,i) \leftrightarrow q_{\text{from}}(\pi,j)) \wedge \bigwedge_{c \in \{1,2\}} (\texttt{to}_c(\pi,i) \leftrightarrow \texttt{to}_c(\pi,j)) \wedge \bigwedge_{op \in \{\texttt{inc},\texttt{dec},\texttt{isZero}\}} (op(\pi,i) \leftrightarrow op(\pi,j)) \tag{13}$$

- transition matches a transition from Δ:

$$\texttt{validTr}(\Delta,\pi,i) \stackrel{\text{def}}{=} \bigvee_{(q,c,op,q') \in \Delta} (q_{\text{from}}(\pi,i)) \wedge \texttt{to}_c(\pi,i)) \wedge op(\pi,i)) \wedge q'_{\text{to}}(\pi,i))) \tag{14}$$

- for times i and j the trace π defines the same unique state:

$$\texttt{sameState}(Q,\pi,i,j) \stackrel{\text{def}}{=} (\bigwedge_{q \in Q} q(\pi,i) \leftrightarrow q(\pi,j)) \wedge \texttt{exactlyOne}(\{q(\pi,i) \mid q \in Q\}) \tag{15}$$

- states in π and π' match the configuration:

$$\texttt{goodStates}(Q,\pi,\pi',i) \stackrel{\text{def}}{=} \bigvee_{q \in Q} (q_{\text{from}}(\pi,i) \leftrightarrow q(\pi,i)) \wedge \bigvee_{q \in Q} (q_{\text{to}}(\pi,i) \leftrightarrow q(\pi',i)) \tag{16}$$

- Only one (the first) guess can affect the evaluation:

$$\texttt{stopMulitpleGuess}(\pi_g,i,j) \stackrel{\text{def}}{=} (\texttt{guess}(\pi_g,i) \vee \texttt{guessed}(\pi_g,i)) \rightarrow \texttt{guessed}(\pi_g,j) \tag{17}$$

We recall that operations update at most one of the counters. To guess for the time point where this change occurs, our encoding uses propositional variables `guess` and `guessed` to be instantiated by an unconstrained trace quantifier. In particular, if either the `guess` is false or `guessed` is true, we require the two traces being compared to have equal values in their counters. We define now how to encode the effect of an operation, where π encodes the transition to be applied as well as the incoming configuration, π' encodes the outgoing configuration and π_g the unconstrained trace with the guess:

$$\texttt{op}(\pi,c,\pi',\pi_g,i,i_+,i_-) \stackrel{\text{def}}{=} (\texttt{mem}_{\overline{c}}(\pi,i) \leftrightarrow \texttt{mem}_{\overline{c}}(\pi',i) \wedge \tag{18}$$

$$(\texttt{isZero}(\pi,i) \rightarrow (\neg\texttt{mem}_c(\pi,i) \wedge \neg\texttt{mem}_c(\pi',i))) \wedge \tag{19}$$

$$((\neg\texttt{guess}(\pi_g,i) \vee \texttt{guessed}(\pi_g,i)) \rightarrow (\texttt{mem}_c(\pi,i) \leftrightarrow \texttt{mem}_c(\pi',i))) \wedge \tag{20}$$

$$((\neg\texttt{guessed}(\pi_g,i) \wedge \texttt{guess}(\pi_g,i) \wedge \texttt{inc}(\pi,i)) \rightarrow (\neg\texttt{mem}_c(\pi,i) \wedge \texttt{mem}_c(\pi',i) \wedge \neg\texttt{mem}_c(\pi',i_+)) \wedge \tag{21}$$

$$((\neg\texttt{guessed}(\pi_g,i) \wedge \texttt{guess}(\pi_g,i) \wedge \texttt{dec}(\pi,i)) \rightarrow (\texttt{mem}_c(\pi,i) \wedge \neg\texttt{mem}_c(\pi',i) \wedge \texttt{mem}_c(\pi',i_-)) \tag{22}$$

We include the time just before i, i_-, and just after, i_+, because time quantification will precede the quantification of π' and, so, we may get a different π' at each time point. In the time-prefixed formula defined next, we encode the requirement that all true valuations of $\texttt{mem}_1$ and $\texttt{mem}_2$ must be consecutive, starting from the beginning of the trace. Hence, by

using i_- and i_+ we ensure that the change around time i is consistent with the operation leading to it. We combine all requirements above in the following time-prefixed hypertrace logic formula $\varphi_{\mathcal{M}}$ for a given machine $\mathcal{M}=(Q,\Delta,\hat{q})$:

$$\exists i_0 \forall i\ \exists i_+\ \exists i_-\ \forall j\ \forall \pi{::}T\ \exists \pi'{::}T\ \exists \pi_{\hat{q}}{::}T\ \exists \pi_g \big(\tag{23}$$

$$(i_0 \le j \wedge i_- \le i \wedge i < i_+ \wedge (i<j \rightarrow i_+ \le j) \wedge (i_- = i \rightarrow i = i_0) \wedge (j<i \rightarrow j \le i_-)) \rightarrow \tag{24}$$

$$(\mathtt{mem}_1(\pi,i_+) \rightarrow \mathtt{mem}_1(\pi,i)) \wedge (\mathtt{mem}_2(\pi,i_+) \rightarrow \mathtt{mem}_2(\pi,i)) \wedge \tag{25}$$

$$\hat{q}(\pi_{\hat{q}},i) \wedge (\hat{q}(\pi,i) \rightarrow (\neg \mathtt{mem}_1(\pi,i) \wedge \neg \mathtt{mem}_2(\pi,i))) \wedge \tag{26}$$

$$\mathtt{stopMulitpleGuess}(\pi_g,i,i_+) \wedge \tag{27}$$

$$\mathtt{singleTr}(Q,\pi,i) \wedge \mathtt{sameTr}(Q,\pi,i,i_+) \wedge \mathtt{validTr}(\Delta,\pi,i) \wedge \tag{28}$$

$$\mathtt{sameState}(Q,\pi,i,i_+) \wedge \mathtt{goodStates}(Q,\pi,\pi',i) \wedge \tag{29}$$

$$(\mathtt{to}_1(\pi,i) \rightarrow \mathtt{op}(\pi,1,\pi',\pi_g,i,i_+,i_-)) \wedge (\mathtt{to}_2(\pi,i) \rightarrow \mathtt{op}(\pi,2,\pi',\pi_g,i,i_+,i_-))\big) \tag{30}$$

A set of traces $\mathrm{T} \models_{\mathbb{T}} \varphi_{\mathcal{M}}$ satisfies the following properties, where for $\mathcal{Y} \subseteq \mathcal{X}$ we define its projection over a trace as $\tau|_{\mathcal{Y}} = (\tau[0] \cap \mathcal{Y})(\tau[1] \cap \mathcal{Y}) \ldots$:

- exists $\tau \in \mathrm{T}$ and $(\hat{q},c,op,q') \in \Delta$ s.t. $\tau|_{\{\hat{q},\hat{q}_{\text{from}},\mathtt{to}_c,op,q'_{\text{to}},\mathtt{mem}_1,\mathtt{mem}_2\}} = \{\hat{q},\hat{q}_{\text{from}},\mathtt{to}_c,op,q'_{\text{to}}\}^{\omega}$ encoding the initial state where both counters are 0;
- for all $\tau \in \mathrm{T}$, $\tau|_{\{\mathtt{mem}_1\}} = \{\mathtt{mem}_1\}^{n_1}\{\}^{\omega}$, $\tau|_{\{\mathtt{mem}_2\}} = \{\mathtt{mem}_2\}^{n_2}\{\}^{\omega}$ and there exists $q \in Q$ and $(q,c,op,q') \in \Delta$ s.t. $\tau|_{\{q,q_{\text{from}},\mathtt{to}_c,op,q'_{\text{to}}\}} = \{q,q_{\text{from}},\mathtt{to}_c,op,q'_{\text{to}}\}^{\omega}$, which are unique and denoted by $\text{config}(\tau)=(q,n_1,n_2)$ and $\text{trans}(\tau)=(q,c,op,q')$;
- for each trace assigned to π, there can be only one time point where for the unconstrained trace assigned to π_g the value of `guess` is true and `guessed` is false;
- the trace assigned to π' encodes a correct next state from the transition defined by π.

$\mathrm{T} \models_{\mathbb{T}} \varphi_{\mathcal{M}}$, iff T has a trace encoding the initial state of $\mathcal{M}$ and, starting from that trace we can define a valid infinite computation of $\mathcal{M}$ using only traces in T. ◀

7 Related Work

The first work to explore the connection between hyperlogics (specifically HyperLTL) and classical first-order reasoning is by Finkbeiner and Zimmermann [8]. They extend FO[<] with the *equal-level* predicate E, which allows comparisons between positions at the same time point across different traces. They also use the bounded-promptness (also referred to as bounded-termination) property from [4] to show that $FO[<,E]$ is strictly more expressive than HyperLTL. Later, Bartocci et al. introduce hypertrace logic [1], which takes a different approach by extending FO[<] to a two-sorted logic, explicitly distinguishing between sorts for time and (constrained) traces. They prove that $FO[<,E]$ and hypertrace logic (with only constrained trace quantifiers) are equivalent. More recently, Beutner and Finkbeiner, in [2], presented a tool to check the satisfiability of hyperproperties specified in HyperLTL by translating them to first-order logic. Their translation follows the same approach as hypertrace logic: it considers separate sorts for traces and time. Other works have explore reasoning about hyperproperties using second-order quantification. For example, Hyper^{2}LTL [3] extends HyperLTL by introducing second-order quantification over the set of all possible traces, in addition to first-order quantification over a given trace set. These approaches fall outside the scope of our work, as we focus on first-order definable languages.

The boundaries for the decidability of the satisfiability problem of HyperLTL were first investigated in [6]. These results were extended, in [14], to understand further which HyperLTL fragments have a decidable satisfiability checking. In [5], Coenen et al. give a complete picture of the hierarchy of hyperlogics based on extensions of temporal logics. In particular, they

compare and summarize results for HyperLTL and HyperQPTL. A different line of work looks into LTL interpretation with team semantics, called TeamLTL. Team semantics generalizes classical first-order logic by evaluating formulas over sets of assignments, called teams, rather than single assignments. TeamLTL, based on this framework, does not use explicit trace quantifiers. Instead, it adopts the modular approach of team semantics, where reasoning over multiple traces is introduced implicitly through the use of generalized atoms. In [20], the authors establish the relation between different extensions of TeamLTL to fragments of HyperQPTL$^+$, which generalizes HyperQPTL to include non-uniform quantification over propositional variables.

8 Conclusion

In this work, we introduced an extension of hypertrace logic with unconstrained trace quantifiers, effectively extending its expressive power. We established expressiveness results connecting fragments of hypertrace logic to well-known temporal logics: unconstrained hypertrace logic is equivalent to S1S, while trace-prefixed hypertrace logic corresponds to HyperQPTL. We also identified a decidable fragment of the logic, where constrained trace quantifier alternation is restricted to a single existential-to-universal switch. This generalizes a known decidability result for HyperQPTL by allowing arbitrary placements of time quantifiers. Finally, we identified a fragment of time-prefixed hypertrace logic with undecidable satisfiability checking. For future work, it remains open how the trace- and time-prefixed fragments relate in terms of expressiveness. Additionally, we plan to explore the connection between hyperlogics and classical first-order reasoning to enable the transfer of techniques and results across these areas.

References

1 Ezio Bartocci, Thomas Ferrère, Thomas A. Henzinger, Dejan Nickovic, and Ana Oliveira da Costa. Flavors of sequential information flow. In Bernd Finkbeiner and Thomas Wies, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 1–19. Springer International Publishing, 2022. doi:10.1007/978-3-030-94583-1_1.

2 Raven Beutner and Bernd Finkbeiner. Checking satisfiability of hyperproperties using first-order logic. In S. Akshay, Aina Niemetz, and Sriram Sankaranarayanan, editors, *Automated Technology for Verification and Analysis (ATVA)*, pages 198–211, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-78750-8_10.

3 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-order hyperproperties. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification (CAV)*, pages 309–332, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-37703-7_15.

4 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *International Conference on Foundations of Software Science and Computation Structures*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

5 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785713.

6 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2016.13.

7 Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing ω-regular hyperproperties. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification (CAV)*, pages 40–63, Cham, 2020. Springer International Publishing.

8 Bernd Finkbeiner and Martin Zimmermann. The First-Order Logic of Hyperproperties. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2017.30.

9 Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 163–173, 1980. doi:10.1145/567446.567462.

10 Christopher Hahn. *Logical and deep learning methods for temporal reasoning*. PhD thesis, Saarland University, 2021. doi:10.22028/D291-35192.

11 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, July 1990. doi:10.1145/78969.78972.

12 Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.

13 Yonit Kesten and Amir Pnueli. Complete proof system for qptl. *Journal of Logic and Computation*, 12(5):701–745, October 2002. doi:10.1093/logcom/12.5.701.

14 Corto Mascle and Martin Zimmermann. The Keys to Decidable HyperLTL Satisfiability: Small Models or Very Simple Formulas. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2020.29.

15 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., USA, 1967.

16 Amir Pnueli. The temporal logic of programs. In *Proc. of FOCS77: the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

17 Markus N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016. doi:10.22028/D291-26650.

18 Gaëtan Regaud and Martin Zimmermann. The complexity of hyperqptl. *arXiv preprint arXiv:2412.07341*, 2024. doi:10.48550/arXiv.2412.07341.

19 J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Logic, Methodology and Philosophy of Science*, volume 44 of *Studies in Logic and the Foundations of Mathematics*, pages 1–11. Elsevier, 1960. doi:10.1016/S0049-237X(09)70564-6.

20 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In Mikołaj Bojańczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 52:1–52:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2021.52.

Overlay Network Construction: Improved Overall and Node-Wise Message Complexity

Yi-Jun Chang
National University of Singapore, Singapore

Yanyu Chen
National University of Singapore, Singapore

Gopinath Mishra
National University of Singapore, Singapore

Abstract

We consider the problem of constructing distributed overlay networks, where nodes in a reconfigurable system can create or sever connections with nodes whose identifiers they know. Initially, each node knows only its own and its neighbors' identifiers, forming a local channel, while the evolving structure is termed the global channel. The goal is to reconfigure any connected graph into a desired topology, such as a bounded-degree expander graph or a well-formed tree (WFT) with a constant maximum degree and logarithmic diameter, minimizing the total number of rounds and message complexity. This problem mirrors real-world peer-to-peer network construction, where creating robust and efficient systems is desired.

We study the overlay reconstruction problem in a network of n nodes in two models: GOSSIP-reply and HYBRID. In the GOSSIP-reply model, each node can send a message and receive a corresponding reply message in one round. In the HYBRID model, a node can send $O(1)$ messages to each neighbor in the local channel and a total of $O(\log n)$ messages in the global channel.

In both models, we propose protocols for WFT construction with $O(n \log n)$ message complexities using messages of $O(\log n)$ bits. In the GOSSIP-reply model, our protocol takes $O(\log n)$ rounds while in the HYBRID model, our protocol takes $O(\log^2 n)$ rounds. Both protocols use $O\left(n \log^2 n\right)$ bits of communication.

We obtain improved bounds over prior work:

GOSSIP-reply: A recent result by Dufoulon et al. (ITCS 2024) achieved $O\left(\log^5 n\right)$ round complexity and $O\left(n \log^5 n\right)$ message complexity using messages of at least $\Omega\left(\log^2 n\right)$ bits in GOSSIP-reply. With messages of size $O(\log n)$, our protocol achieves an optimal round complexity of $O(\log n)$ and an improved message complexity of $O(n \log n)$.

HYBRID: Götte et al. (Distributed Computing 2023) showed an optimal $O(\log n)$-round algorithm with $O(\log^2 n)$ global messages per round which incurs a message complexity of $\Omega(m)$, where m is the number of edges in the initial topology. At the cost of increasing the round complexity to $O(\log^2 n)$ while using only $O(\log n)$ messages globally, our protocol achieves a message complexity that is independent of m. Our approach ensures that the total number of messages for node v, with degree $\deg(v)$ in the initial topology, is bounded by $O(\deg(v) + \log n)$, while the algorithm of Götte et al. requires $O\left(\deg(v) + \frac{\log^4 n}{\log \log n}\right)$ messages per node.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, Overlay networks, Expander graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.21

Related Version *Full Version*: https://arxiv.org/abs/2412.04771 [14]

Funding The research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (24-1323-A0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.



© Yi-Jun Chang, Yanyu Chen, and Gopinath Mishra;
licensed under Creative Commons License CC-BY 4.0



Acknowledgements We are grateful to an anonymous FSTTCS reviewer for suggesting the use of the hash functions of [31] to reduce the number of bits required for identifying an outgoing edge. In an earlier version of this work, this task had been carried out using the linear sketches of [1]. The incorporation of the reviewer's suggestion results in an $O(\log n)$-factor improvement in the overall communication complexity.

1 Introduction

Many of today's large-scale distributed systems on the Internet, such as peer-to-peer (P2P) and overlay networks, prioritize forming logical networks over relying (only) on the physical infrastructure of the underlying network. In these systems, direct connections between nodes can be virtual, using the physical connections of the Internet, and nodes are considered connected if they know each other's IP addresses, allowing them to communicate and establish links. Examples of such systems include cryptocurrencies, the Internet of Things, the Tor network, and overlay networks like Chord [43], Pastry [42], and skip graphs [4]. These networks have the flexibility to reconfigure themselves by choosing which connections to establish or drop. This work focuses on the challenge of efficiently constructing a desired overlay network from any starting configuration of n nodes, recognizing that the problem has a lower bound of $O(\log n)$, since even in an optimal scenario, it takes at least $O(\log n)$ rounds for two endpoints to connect if the nodes initially form a line.

In this paper, we address the well-explored challenge of efficiently constructing overlay topologies in a distributed manner within reconfigurable networks. This task is crucial in modern P2P networks, where topological properties are vital in ensuring optimal performance. Over the past two decades, numerous theoretical studies [39, 34, 23, 17, 29, 10], have focused on developing P2P networks that exhibit desirable characteristics such as high conductance, low diameter, and resilience to substantial adversarial deletions. The common approach in these studies is to build a bounded-degree random graph topology in a distributed way, which ensures these properties. This approach leverages the fact that random graphs are likely to be expanders, possessing all the desired attributes [37, 27]. Random graphs have been extensively used to model P2P networks [39, 34, 37, 35, 17, 11, 8, 7, 9], and the random connectivity topology has been widely adopted in many contemporary P2P systems, including those underpinning blockchains and cryptocurrencies like Bitcoin [36].

Several works have focused on overlay construction that transforms an arbitrary connected graph into a desired topology [3, 24, 25, 26, 19]. While minimizing the number of rounds is the primary objective, reducing the total number of messages exchanged (message complexity) is also crucial. The message complexity in [19] is $O(n \log^5 n)$, whereas in other works it is $\Omega(m)$, where n is the number of nodes and m is the number of edges in the initial topology. In this work, we propose protocols in two models, GOSSIP-reply and HYBRID (defined formally later), that are both round-efficient and communication-efficient. Additionally, our protocol in the HYBRID model optimizes the node-wise message complexity (i.e., the number of messages each node sends and receives based on its degree) compared to the previous work [26].

Before discussing our results and comparing them with previous works, we formally introduce the models in the next section.

1.1 Models

We consider synchronous models on a fixed set of nodes V, where each node $v \in V$ has a unique identifier $\mathrm{id}(v)$ of length $O(\log n)$, with $n = |V|$. The *local/input* network is represented by a graph $G = (V, E)$. Without loss of generality, we assume that G is

connected. Computation proceeds in synchronous rounds, during which the *global/overlay* network evolves. In each round, nodes can send and receive messages and perform local computations. Unless otherwise specified, messages are $O(\log n)$ bits.

The network is *reconfigurable* in the sense that if u knows the identifier of v, then u can send a reconfiguration message to v to establish or drop the communication link. Lastly, we also allow implicit edge deletion since it can be easily implemented by only keeping edges established after round r.

In general, we assume that the communication links are reliable, and no messages are dropped when the message capacity of the link is not exceeded. We also assume that there is sufficient memory on each computing node for our algorithm to run correctly and is capable of processing all incoming messages at the start of a round within that same round.

In this paper, as already mentioned, we consider two synchronous models: GOSSIP-reply and HYBRID, formally defined as follows.

GOSSIP-reply model. One of the earliest works in overlay construction by Angluin et al. [3] considered a model – now known as the GOSSIP model – where each node is allowed to send only one message per round. Recently, a reply version of this model, the GOSSIP-reply (b) model, was introduced by [19], where they developed the first communication-efficient protocol.[1] In this model, each node v can perform the following actions in one round:

1. Send a message of $O(b)$ bits to a neighbor, where any node whose identifier is known to v is considered a neighbor of v. We call this message the *contacting message*.
2. Receive all messages sent to v. Do some local computation.[2]
3. Send an $O(b)$-bit reply to each of the contacting messages.
4. Receive an $O(b)$-bit reply. Do some local computation.

Observe that in this model, there will be at most $2n$ messages sent in each round, n contacting messages and n replying messages. Hence, any algorithm with $O(T)$ round complexity has $O(nT)$ message complexity.

HYBRID model. The hybrid model was proposed in [6] to study shortest path problems and later considered by [26] in the context of overlay construction problem. In this model, the communication is done over both local and global channels. The HYBRID (α, β, γ) model is defined by three parameters α, β, and γ:

- Each message size is $O(\alpha)$ bits.
- Each node can send and receive $O(\beta)$ messages per round to each local neighbor, i.e., the local capacity is $O(\beta)$.
- Each node can send and receive $O(\gamma)$ messages per round to any node whose identifier it knows, i.e., the global capacity is $O(\gamma)$.[3]

A subtle aspect of the model arises when a node is sent more messages in a round than its capacity permits. A standard assumption is that the node receives an arbitrary subset of these messages while the rest are dropped. All our algorithms guarantee that, with high probability, this situation never occurs, so the exact handling of such cases is irrelevant to our results.

[1] In [19], it is referred to simply as the GOSSIP-based model or the P2P-GOSSIP model.

[2] Although local computation in Step 2 is not explicitly included in the original definition of [19], preparing the outgoing messages appears to require some. Nevertheless, two rounds of the model without local computation in Step 2 suffice to simulate one round of the model with it.

[3] All $O(\gamma)$ messages can be directed to a single global neighbor or spread across $O(\gamma)$ global neighbors.

In this work, we study the HYBRID$(\log n, 1, \log n)$ model, where each message has size $O(\log n)$ bits, the local capacity is $O(1)$, and the global capacity is $O(\log n)$. Importantly, a node may send a number of local messages proportional to its degree, whereas in the global network it is limited to $O(\log n)$ messages. The rationale for assigning different capacities lies in the cost assumption: local communications, which take place on the given topology, are cheaper, while global communications, which require establishing new links, are more costly.

In addition to the GOSSIP-reply and HYBRID models, the P2P-CONGEST model has also been widely studied [24, 25, 26]. In P2P-CONGEST, each node can send and receive up to $O(\Delta \log n)$ messages of $O(\log n)$ bits per round, where Δ is the maximum degree of any node in the initial topology. It is important to note that HYBRID $(\log n, 1, \log n)$ is *weaker* than P2P-CONGEST in the sense that any protocol in HYBRID $(\log n, 1, \log n)$ can be simulated in the P2P-CONGEST model without asymptotically increasing the round or message complexity asymptotically. Therefore, all results obtained in HYBRID $(\log n, 1, \log n)$ also apply to P2P-CONGEST.

1.2 Our contribution and comparison with prior work

As already mentioned, we focus on designing algorithms in both the GOSSIP-reply and HYBRID models that are efficient in terms of rounds and communication. Additionally, we demonstrate that our protocol for the HYBRID model also achieves improved node-wise message complexity. We also discuss implications for the P2P-CONGEST model. See Appendix B for a discussion on the tradeoffs between different complexity measures and the motivation for studying node-wise message complexity.

Unless otherwise specified explicitly, all of our results including prior works are randomized and succeed *with high probability* (w.h.p.), i.e., with probability at least $1 - 1/\text{poly}(n)$.

1.2.1 Our result in the GOSSIP-reply model

Our result in the GOSSIP-reply model is summarized in the following theorem. It shows that starting from any arbitrary topology, we can transform it into a star overlay. We then demonstrate how a star overlay can be converted into a desired topology by leveraging the properties of the GOSSIP-reply model.

▶ **Theorem 1.1.** *There is a protocol in the* GOSSIP-reply (b) *model that can construct a star overlay in* $O\left(\log n \cdot \max\left(\frac{\log n}{b}, 1\right)\right)$ *rounds with* $O\left(n \log n \cdot \max\left(\frac{\log n}{b}, 1\right)\right)$ *messages w.h.p.*

Note that building a star topology is similar to doing leader election in the *reconfigurable* network. Observe that when the star topology is constructed, we can treat the distinguished center node in the star as the leader and perform many tasks on the leader node locally. More specifically, in GOSSIP-reply(b), we can reconfigure the network from the star topology to any topology whose maximum degree is $\Delta = O\left(\frac{b}{\log n}\right)$ in $O(1)$ round.

▶ **Observation 1.2.** *If the initial topology G is a star, then there is an $O(\Delta(H))$-round protocol in the* GOSSIP-reply (b) *model to construct an overlay network with a desired topology H whose maximum degree is* $\Delta(H) = O\left(\frac{b}{\log n}\right)$.

Proof. Firstly, every node except the distinguished center node sends a request message to the center node. Then the center node will compute an assignment of the nodes in the desired topology locally and reply to every node v with their neighborhood $N_H(v)$. Each node then takes $O(\Delta(H))$ rounds to establish connections with the new neighbors formally. Note that

the center node needs to send $O(\Delta(H)\log n)$ bits to every leaf node. Since $\Delta(H) = O\left(\frac{b}{\log n}\right)$, the information that the center node needs to send is $O(\Delta(H)\log n) = O(b)$ bits, which can be sent in one message. ◀

We obtain the following corollary by applying Observation 1.2 and Theorem 1.1 with $b = O(\log n)$.

▶ **Corollary 1.3.** *There is a protocol in the* GOSSIP-reply $(\log n)$ *model that can construct any constant degree overlay network in* $O(\log n)$ *rounds with* $O(n \log n)$ *messages w.h.p.*

Comparison with Dufoulon et al. [19]. The algorithm by Dufoulon et al. in the GOSSIP-reply (b) model, with $b = \Omega(\log^2 n)$, converts any arbitrary topology into a constant-degree expander in $O(\log^5 n)$ rounds, with a message complexity of $O(n \log^5 n)$. Thus, Corollary 1.3 provides a strict improvement over [19] in both round and message complexity. Additionally, our algorithm can produce any constant-degree overlay, whereas the algorithm of [19] could only construct a constant-degree expander. The comparison of our result in the GOSSIP-reply (b) model with that of [19] is also presented in Table 1.

Table 1 Improvements in the GOSSIP-reply (b) model.

Reference	b	Rounds	Total message complexity	Target topology
[19]	$\Omega(\log^2 n)$	$O(\log^5 n)$	$O(n \log^5 n)$	$O(1)$-degree expander
Corollary 1.3	$O(\log n)$	$O(\log n)$	$O(n \log n)$	Any $O(1)$-degree graph

1.2.2 Our result in the HYBRID model

In the HYBRID $(\log n, 1, \log n)$ model, our main result is summarized in the following theorem: we show that starting from any arbitrary initial topology, it is possible to transform the network into a *well-formed tree* (WFT) efficiently. A well-formed tree with n nodes is defined as one that has a constant maximum degree and a depth of $O(\log n)$. Furthermore, we discuss how a well-formed tree can be efficiently converted into a constant-degree expander in HYBRID$(\log n, 1, \log n)$ model using the results from prior work in [19, 26].

▶ **Theorem 1.4.** *There is a protocol in the* HYBRID $(\log n, 1, \log n)$ *model that can construct a well-formed tree overlay from any input graph* G *in* $O\left(\log^2 n\right)$ *rounds with* $O(n \log n)$ *messages w.h.p. Moreover, each node* v *sends and receives at most* $O(\deg_G(v) + \log n)$ *messages throughout the protocol.*

We remark that, although the sum of the node-wise bounds appears to imply an $O(m)$ message complexity, in our algorithm only a small subset of nodes may incur as many as $\Omega(\deg_G(v))$ messages, and the message complexity remains bounded by $O(n \log n)$. Due to the use of randomness, it is not possible to determine in advance which nodes incur these higher costs.

Our algorithm follows a Boruvka-style cluster-merging process while maintaining the invariant that each cluster induces a well-formed tree. Outgoing edges are identified using sketching techniques. To achieve the node-wise message bound of $O(\deg_G(v) + \log n)$, we address the high communication load on star centers during cluster merges by introducing a matching-based method that pairs clusters for merging, even without a direct connecting edge, while ensuring that the total number of clusters reduces by a constant factor in each round of the merging process.

To further optimize the message complexity, we employ a randomized procedure for constructing an $O(\log n)$-degree, $O(\log n)$-depth tree from a cycle. This improves upon the deterministic pointer-jumping process of prior work, yielding an $O(\log n)$-factor reduction in message cost.

Through minor modifications of the works in [19, 26], we restate the following lemma, which enables the efficient transformation of a constant degree overlay network (e.g., a well-formed tree) into a constant degree expander network with constant conductance w.h.p. in the $\mathsf{HYBRID}(\log n, 1, \log n)$ model.

▶ **Lemma 1.5** ([19, 26]). *Consider the* $\mathsf{HYBRID}(\log n, 1, \log n)$ *model. For any constant* $\Phi \in (0, 1/10]$*, there is a protocol that takes* $O\left(\log^2 n\right)$ *rounds and* $O\left(n \log^2 n\right)$ *messages to convert an* $O(1)$*-degree overlay network into an* $O(1)$*-degree expander network with conductance at least* Φ*, w.h.p. Moreover, each node* v *sends and receives at most* $O\left(\frac{\log^3 n}{\log \log n}\right)$ *messages w.h.p.*

By applying Lemma 1.5 after Theorem 1.4, we can construct an expander overlay network in the $\mathsf{HYBRID}(\log n, 1, \log n)$ model in $O(\log^2 n)$ rounds with $O(n \log^2 n)$ messages.

▶ **Corollary 1.6.** *There is a protocol in the* $\mathsf{HYBRID}(\log n, 1, \log n)$ *model with the following guarantees:*

- *It constructs a constant-degree expander graph from any input graph* G *in* $O\left(\log^2 n\right)$ *rounds using* $O\left(n \log^2 n\right)$ *messages w.h.p.*
- *Each node* v *sends and receives at most* $O\left(\deg_G(v) + \frac{\log^3 n}{\log \log n}\right)$ *messages.*

Comparison with Götte et al. [26]. Götte et al. investigated the problem of overlay reconstruction in the $\mathsf{HYBRID}(\log n, 1, \log^2 n)$ model, aiming to convert an arbitrary initial topology into a well-formed tree or a constant-degree expander. Their algorithm achieved an optimal round complexity of $O(\log n)$ rounds with a message complexity of $\Omega(m + n \log^3 n)$, and the maximum number of messages sent or received by a node of degree $\deg_G(v)$ is $O\left(\deg_G(v) + \frac{\log^4 n}{\log \log n}\right)$. In comparison, although our result in Theorem 1.4 and Corollary 1.6 require $O(\log^2 n)$ rounds, it operates in the weaker $\mathsf{HYBRID}(\log n, 1, \log n)$ model. Crucially, the message complexity of our algorithm does not depend on m, and it achieves better node-wise message complexity compared to [26]. It is important to note, however, that our approach does not lead to an $O(\log n)$-round algorithm even in the $\mathsf{HYBRID}(\log n, 1, \log^2 n)$ model. The comparison of our result in the HYBRID model with that of [26] is presented in Table 2. An open question remains: is it possible to achieve the optimal $O(\log n)$ rounds in the $\mathsf{HYBRID}(\log n, 1, \log n)$ model (or even in the $\mathsf{HYBRID}(\log n, 1, \log^2 n)$ model) with a message complexity of $O(n \cdot \text{poly}(\log n))$?

Table 2 Improvements in the $\mathsf{HYBRID}\,(\log n, 1, \gamma)$ model.

Reference	γ	Rounds	Message complexity		Target topology
			Total	Node-wise	
Götte et al. [26]	$O(\log^2 n)$	$O(\log n)$	$\Omega\big(m + n \log^3 n\big)$	$O\big(\deg_G(v) + \frac{\log^4 n}{\log \log n}\big)$	WFT / $O(1)$-degree expander
Theorem 1.4	$O(\log n)$	$O(\log^2 n)$	$O(n \log^2 n)$	$O\big(\deg_G(v) + \log^2 n\big)$	WFT
Corollary 1.6	$O(\log n)$	$O(\log^2 n)$	$O(n \log^2 n)$	$O\big(\deg_G(v) + \frac{\log^3 n}{\log \log n}\big)$	$O(1)$-degree expander

1.2.3 Implication in P2P-CONGEST model

Götte et al. [26] considered two models: P2P-CONGEST and HYBRID $(\log n, 1, \log^2 n)$, which are not directly comparable. In particular, the result of Götte et al. [26] in HYBRID $(\log n, 1, \log^2 n)$ does not translate directly to P2P-CONGEST. However, as already mentioned before, any protocol in HYBRID $(\log n, 1, \log n)$ can be simulated in the P2P-CONGEST model without asymptotically increasing the round or message complexity. Therefore, from Theorem 1.4 and Corollary 1.6, we obtain the following corollary.

▶ **Corollary 1.7.** *There is a protocol in the* P2P-CONGEST *model that can construct a well-formed tree or a constant-degree expander graph from any input graph G in $O(\log^2 n)$ rounds. The algorithm uses $O(n \log n)$ messages or $O(n \log^2 n)$ w.h.p. for well-formed tree or a constant-degree expander graph, respectively. Moreover, each node v sends and receives at most $O(\deg_G(v) + \log n)$ or $O\left(\deg_G(v) + \frac{\log^3 n}{\log \log n}\right)$ messages depending on whether the target topology is a well-formed tree or a constant-degree expander graph, respectively.*

Comparison with Götte et al. [26]. Götte et al. studied the overlay reconstruction problem in the P2P-CONGEST model, where the goal was to transform an arbitrary initial topology into a well-formed tree. Their solution achieved an optimal round complexity of $O(\log n)$ with a message complexity of $\Omega(m \log n + n \log^2 n)$, and the maximum number of messages sent or received by a node of degree $\deg_G(v)$ is $O\left(\deg_G(v) \cdot \frac{\log^3 n}{\log \log n}\right)$. In contrast, our algorithm, as stated in Corollary 1.7, requires $O(\log^2 n)$ rounds but notably achieves message complexity independent of m and improved node-wise message complexity. The comparison of our implication in the P2P-CONGEST model with that of the result of [26] is also presented in Table 3. A key remaining open question is whether it is possible to attain the optimal $O(\log n)$ rounds in the P2P-CONGEST model with a message complexity of $O(n \cdot \text{poly}(\log n))$.

Table 3 Improvements in P2P-CONGEST model.

Reference	Rounds	Message complexity		Target topology
		Total	Node-wise	
[26]	$O(\log n)$	$\Theta(m \log^2 n)$	$O\left(\deg_G(v) \cdot \frac{\log^3 n}{\log \log n}\right)$	WFT/ $O(1)$-degree expander
Corollary 1.7	$O(\log^2 n)$	$O(n \log n)$	$O(\deg_G(v) + \log n)$	WFT
Corollary 1.7	$O(\log^2 n)$	$O(n \log^2 n)$	$O\left(\deg_G(v) + \frac{\log^3 n}{\log \log n}\right)$	$O(1)$-degree expander

1.3 Number of bits communicated

For all of Theorem 1.1, Corollary 1.3, Theorem 1.4, and Corollary 1.6, the total number of bits communicated among the nodes in all protocols is $O(n \log^2 n)$, due to the use of the hashing techniques from King King, Kutten, and Thorup [31]. This bound on the communication complexity breaks the $\Omega(n \log^3 n)$ barrier of the linear sketch of Ahn, Guha, and McGregor [1] which was used in the previous work [19].

It has been shown that $\Omega(n \log^3 n)$ bits of communication are indeed necessary for several applications of the linear sketch of Ahn, Guha, and McGregor [1], such as distributed and sketching spanning forest [38] and connectivity [44]. More concretely, in the distributed

sketching model, the goal of the connectivity problem is to determine whether an n-node graph G is connected, with each of the n players having access to the neighborhood of a single vertex. Each player sends a message to a central referee, who then decides whether G is connected. Yu [44] established that for the referee to decide correctly with probability 1/4, the total communication must be at least $\Omega(n \log^3 n)$ bits.

The ability to break this barrier stems from being able to communicate in both ways in multiple rounds as opposed to the one-round one-way setting described above. The optimality of the hashing technique from King, Kutten, and Thorup [31] is much less well-known, and it remains open whether their communication complexity bound can be further improved. Any such improvement will likely lead to improvements in the $O(n \log^2 n)$ communication complexity bound in this paper as well as many other applications of the hashing technique.

1.4 Organization

In Section 2, we present the basic graph terminologies and tools. In Section 3, we present our protocol in the GOSSIP-reply model, proving Theorem 1.1. In Section 4, we present our protocol in the HYBRID model, proving Theorem 1.4. In Appendix A, we present a comprehensive discussion of related work. In Appendix B, we discuss the tradeoffs between some complexity measures. For any missing details, please refer to the full version of the paper [14].

2 Preliminaries

A graph is defined as $G = (V, E)$, where $E \subseteq \binom{V}{2}$, as the edges are undirected. The graph does not allow self-loops or multi-edges. Let $n = |V|$ and $m = |E|$. The neighborhood of a vertex v in G is denoted as $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$, and the degree of a vertex v in G is defined as $\deg_G(v) := |N_G(v)|$. The maximum degree of the graph is represented as $\Delta(G) = \max_{v \in V} \deg_G(v)$. The distance $d(u, v)$ between any two nodes u and v is the number of edges in the shortest path between them. The diameter of a graph is the maximum distance between any two nodes in the graph. The set of connected components of G is denoted as $\mathsf{CC}(G)$, and the number of connected components in G is represented by $\mathsf{cc}(G)$.

A star graph, denoted as $K_{1,n-1} = (V, E)$, has a distinguished node $v \in V$ such that an edge $e = \{u, w\} \in E$ exists if and only if $v \in e$. A tree T is a connected acyclic graph. A rooted tree T_v is a tree with a distinguished node v serving as the root. The depth of a rooted tree T_v is the maximum distance from the root v to any other node in the tree. A *well-formed tree* is defined as a rooted tree with a constant maximum degree and a depth of $O(\log n)$. A *satisfactory tree* is defined as a rooted tree with $O(\log n)$ maximum degree and $O(\log n)$ depth.

We assume that each node has an ID of length $O(\log n)$ bits. The ID of an edge is a concatenation of the node IDs with the smaller first. We use $\eta(T_x)$ to denote the maximum edge ID among all edges incident to nodes in T_x.

Expander. The volume of any subset $S \subseteq V$ is defined as $\text{vol}(S) := \sum_{v \in S} \deg_G(v)$. The conductance of a subset $S \subseteq V$, where $|S| \neq 0$ and $|S| \neq |V|$, is given by

$$\Phi_G(S) := \frac{|E(S, V \setminus S)|}{\min(\text{vol}(S), \text{vol}(V \setminus S))},$$

where $E(S, V \setminus S) := \{\{u, v\} \in E \mid u \in S, v \in V \setminus S\}$ represents the set of edges between S and its complement.

The conductance of the graph G is defined as

$$\Phi(G) := \min_{S \subseteq V, S \neq \emptyset, S \neq V} \Phi_G(S).$$

Informally, a graph is considered an expander if it has high conductance. The thresholds commonly used to define high conductance vary by context, including $1/n^{o(1)}$, $1/\operatorname{polylog}(n)$, and $1/O(1)$. In this paper, we define an expander as a graph with conductance of $1/O(1)$.

Broadcast-and-echo. A basic distributed protocol to disseminate and gather information is *broadcast-and-echo*. It is initiated by some node x and messages are relayed in a BFS manner, with possible modifications to the messages down the broadcasting tree. Then this process reaches the leaves, leaf nodes echo with some messages back to their parents. Internal nodes wait untill all the messages are gathered before sending a computed aggregated message to their parents. This process takes $O(D(T_x))$ rounds and $O(|T_x|)$ messages, where T_x refers to the broadcasting tree in this process.

More generally, in the CONGEST model with bandwidth B bits, a broadcast-and-echo initiated by x in T_x with a maximum message size of S bits can be done in $O\left(\frac{S}{B} + D(T_x)\right)$ rounds and $O\left(\frac{S}{B}|T_x|\right)$ messages, via message pipelining.

Find any outgoing edge. For any tree T_x rooted at x, we call edges between T_x and $V \setminus T_x$ outgoing. Linear sketch techniques used in previous works by [1, 30, 40, 19] of $O(\log^2 n)$ bits can be used to sample an outgoing edge with constant success probability. To save on message complexity, we instead use a subroutine from [31] to find an arbitrary outgoing edge from T_x.

At a high level, this protocol of [31] uses similar observation to the well-known linear graph sketch that internal edges in a tree will contribute 0 to the sum of degree, or XOR of edge IDs. However, it breaks the well-known linear graph sketch [1] into two phases. First, it uses $O(\log n)$ bits to aggregate the parity of the number of edges that is hashed into each log-scale bracket (1,2,4,8, ...). Then, they show that with constant probability there is one log-scale bracket that has exactly one edge hashed to it. This step corresponds to guessing the suitable sampling probability for exactly one outgoing edge to be sampled. They finally spend another $O(\log n)$ bits to identify the identity of the edge by XORing the edge numbers that are in the identified bracket. This process takes four iterations of broadcast-and-echo with message size $O(\log n)$ bits.

For completeness, we describe the protocol FindOutgoing(x) initiated at node x, which returns an edge leaving T_x with probability at least 1/16. The version described here corresponds to FindAny-C(x) in [31]. FindOutgoing(x) uses another protocol from [31] HPTestOut(x) that returns true with high probability if there is an edge leaving T_x and false otherwise. HPTestOut is always correct if true is returned and uses one broadcast-and-echo with message size $O(\log n)$ bits.

FindOutgoing(x):

1. x initiates HPTestOut(x) in T_x and return $\emptyset$ if HPTestOut returns false.
2. Determine the identity of an edge with the following steps:
   a. x broadcasts a random pairwise independent hash function $h : [1, \eta(T_x)] \to [0, r]$ where $r = 2^w > \sum_{v \in T_x} \deg(v)$ for some w .
   b. each node y hashes the ID of all edges incident to it and compute a $\log r$-bit binary vector $\vec{h}(y)$ such that $\vec{h}_i(y) := |\{e \mid y \in e \wedge h(e) < 2^i\}| \mod 2$.

   c. The vector $\vec{h}(T) := \oplus_{y \in T} \vec{h}(y)$ is computed up the tree, in the broadcast-and-echo return to x. Then x broadcasts $min = \min\{i \mid \vec{h}_i(T) = 1\}$.
   d. Each node y computes $s(y) = \oplus\{e \mid y \in e \wedge h(e) < 2^{min}\}$ and $s(T) = \oplus_{y \in T} s(y)$ is computed up the tree in the broadcast-and-echo and returned to x. Observe that if there is exactly one edge leaving T_x with $h(e) < 2^{min}$, then $s(T)$ is its edge ID.
3. x can perform another broadcast-and-echo to check if $s(T_x)$ is indeed a valid edge ID and return $s(T)$ if the check succeeds and $\emptyset$ if the check fails.

▶ **Lemma 2.1** ([31]). *If there is no edge leaving T_x, then FindOutgoing(x) return $\emptyset$. Otherwise, FindOutgoing(x) returns an edge leaving T_x with probability at least $1/16$, else it returns $\emptyset$. The algorithm uses worst-case $O(D(T_x))$ rounds and $O(|T_x|)$ messages.*

3 Star overlay construction in the GOSSIP-reply model

We begin by describing the high-level approach underlying our algorithm for the GOSSIP-reply model. We draw inspiration from the following existing techniques.

Boruvka-style cluster merging with efficient inter-cluster edge selection. We use a Boruvka-style cluster merging approach used in many prior works [19, 24, 3] while maintaining a simple and useful invariant. We start with each node being a single cluster. In each iteration, we select inter-cluster edges and merge the clusters joined by these edges. By ensuring a constant factor reduction in the number of clusters in each iteration, the process terminates in $O(\log n)$ iterations. The challenge here is how to quickly select an outgoing edge effectively (effectiveness measured by small round or message complexity). We adopt Lemma 2.1 to find an outgoing edge efficiently. This was not previously used in the overlay network construction context and is more efficient than the linear graph sketching technique used by [19].

Selective merging to overcome long chains. Overall our protocol works by sampling an inter-cluster edge from each cluster and merging clusters joined by sampled edges. Merging can be potentially slow due to the large diameter when the inter-cluster edges form a long chain connecting many clusters. Therefore, we break this chain via a simple coin-flipping technique, where each cluster flips a coin and will only accept a request if the coins of the requesting and requested clusters satisfy a specific condition. This symmetry-breaking technique is used extensively in many parallel and distributed works in problems such as parallel list ranking [16] and distributed graph connectivity [21].

Faster and simpler merging. A key difference between our protocol and that of [19] is that we maintain a much simpler and useful invariant (maintaining a star in each cluster) that allows us to aggregate information in a cluster and perform merging among clusters much faster.

Star overlay construction protocol. We describe our protocol MergeStar to construct a star overlay in the GOSSIP-reply $(\log n)$ model which proves Theorem 1.1.

The algorithm proceeds in $O(\log n)$ Boruvka-style phases w.h.p. In each phase, a constant factor of clusters is merged into other clusters to form a cluster for the next phase with constant probability. The algorithm starts with each node being a cluster and maintains the following invariant: at the end of each phase, every node in each cluster S agrees on a leader $l(S)$. This is true initially since each node can be the leader of its own cluster. In other words, this invariant implies that each cluster will keep a star overlay topology.

Denote the given input topology as $G = (V, E)$. Let $G_i = (V, E_i)$ be the topology of the overlay network at the end of phase i. Let $G_0 = (V, \emptyset)$, i.e., we start with each node being an isolated node in the overlay network. We refer to a connected component $C = \left(S, E_i \cap \binom{S}{2}\right) \in \mathsf{CC}(G_i)$ as a cluster in G_i.[4] An outgoing edge from the cluster C is an edge in the input graph $G = (V, E)$ connecting a node in S to a node outside S i.e., $\mathrm{Out}(C) := E_G(S, V \backslash S) := \{\{u, v\} \in E \mid u \in S \text{ and } v \in V \backslash S\}$ is the set of outgoing edges of the cluster C.

Each phase consists of three steps. In phase i, we start with the overlay G_{i-1}.

1. *Sampling Step:* Each cluster $C = \left(S, E_i \cap \binom{S}{2}\right)$ finds an outgoing edge e from $\mathrm{Out}(C)$ and a color $\chi(S) \in \{\mathsf{Blue}, \mathsf{Red}\}$, and then sends a *merging request* containing the sampled color $\chi(S)$ to the external node in the sampled edge.
2. *Grouping Step:* Clusters who received *merging requests* decide on which clusters to merge with based on the color and reply either with an *accepting message* or a *rejecting message*. The purpose of the $\{\mathsf{Blue}, \mathsf{Red}\}$-coloring is to prevent long chains of *merging requests* slowing down the *Merging Step*.
3. *Merging Step:* Clusters that agree on merging will perform this step to merge the clusters. Each node in these clusters must agree on a new leader to maintain the invariant.

Sampling step. In this step, each cluster $C = \left(S, E_i \cap \binom{S}{2}\right)$ needs to sample an outgoing edge uniformly at random with constant probability. It will take $\Omega(\Delta)$ rounds if we let each node check if each neighbor is in the same cluster. We will use the FindOutgoing protocol to reduce communication. Each broadcast-and-echo is replaced by each leave in the star sending one request to the leader $l(S)$ for the broad-casted information. This exploits the replying property of GOSSIP-reply. No random walk or PUSH-style information-spreading like [19] is needed. After the FindOutgoing protocol, the leader finds an outgoing edge with constant probability. Then it sends a *merging request* along the sampled edge to the destination. Note that this step works correctly with probability 1/16 due to Lemma 2.1, as long as $cc(G_i) \geq 2$.

Additionally, to facilitate the grouping step, the leader $l(S)$ independently and uniformly samples a color $\chi(S) \in \{\mathsf{Blue}, \mathsf{Red}\}$ for the cluster S and sends the 1-bit information along with the *merging request*.

Grouping step. The node $v \in S'$ that received the *merging request* will reply with the leader $l(S')$. Then each leader will send the request to other cluster leaders. Since each cluster can only initiate one *merging request*, there will be in total $c(G_i)$ merging requests. Thus, there can be cycles or long chains in the graph, which can affect merging speed. Thus, we need to break these chains. We do this by making each cluster leader accept a request if and only if it is Red and the requesting cluster is Blue.

Merging step. Note that after the Grouping step, the *merging requests* viewed as edges among clusters will form a star with the Red clusters as the centers. We can identify the merged clusters in G_{i+1} with the Red clusters in G_i. Thus, We can maintain the invariant in each cluster in G_{i+1} by letting the leader of the Red clusters become the new leader of the merged clusters in G_{i+1}.

[4] Sometimes we also loosely refer to S as the cluster. This should not cause any confusion since a cluster C in G_i is induced by S.

Each Red leader will reply with its own identifier to the Blue leaders. Each Blue cluster leader then broadcasts this new leader identifier to the Blue cluster members by replying to the cluster members' request. Actions taken by different nodes in the merging step are summarized in Table 4.

Table 4 Summary of actions of different nodes in the Merging Step.

Type in G_i	Actions in the Merging Step
Cluster member	Send leader update requests to its leader in G_i.
Red leader u	Reply to leader update requests with its own identifier.
Blue leader v	Received acceptance decision from u; $\begin{cases} \text{Reply to leader update requests with id}(u) & \text{if } v \text{ is accepted} \\ \text{Reply to leader update requests with id}(v) & \text{if } v \text{ is rejected} \end{cases}$

After this, each node in G_{i+1} will agree on the same leader (i.e., the Red leader), thereby maintaining the invariant.

Analysis. We bound the number of phases that the algorithm takes before it terminates w.h.p.

▶ **Lemma 3.1.** *Let $i \in \mathbb{N}$ such that $\mathsf{cc}(G_i) \geq 2$. We have $\mathbb{E}[\mathsf{cc}(G_{i+1})] \leq \frac{63}{64}\mathsf{cc}(G_i)$.*

Proof. First, observe that in one phase, each cluster has done the sampling step and the grouping step which can affect the number of clusters at the end of the phase.

Let X_C be the indicator random variable for the event that either C fails to sample an outgoing edge or the *requesting message* initiated by C is **rejected**, for each cluster $C \in \mathsf{CC}(G_i)$. As $cc(G_i) \geq 2$, the leader of each cluster $C \in \mathsf{CC}(G_i)$ find an outgoing edge from C with probability at least $1/16$. Moreover, note that the requesting message from cluster C will be accepted with probability $1/4$. So, we have

$$\mathbb{E}[X_C] \leq 1 - \frac{1}{16}\frac{1}{4} = \frac{63}{64}.$$

By linearity of expectation, we have

$$\mathbb{E}\left[\mathsf{cc}(G_{i+1})\right] = \mathbb{E}\left[\sum_{C \in \mathsf{CC}(G_i)} X_C\right] = \sum_{C \in \mathsf{CC}(G_i)} \mathbb{E}\left[X_C\right] \leq \frac{63}{64}\mathsf{cc}(G_i). \qquad \blacktriangleleft$$

▶ **Lemma 3.2.** *The protocol* MergeStar *terminates in $O(\log n)$ phases w.h.p.*

Proof. Let $t = c \log n$ for some sufficiently large constant c that we will determine later. Our goal is to show that $\mathsf{cc}(G_t) = 1$ w.h.p., implying that the protocol terminates in $O(\log n)$ phases w.h.p. Define $Y_i := \mathsf{cc}(G_i) - 1$, where i is a non-negative integer. Initially, we have $Y_0 = n - 1$, and the process terminates in phase i when $Y_i = 0$. We aim to demonstrate that $Y_t = 0$ w.h.p. Observe that $Y_0, \ldots, Y_t$ form a sequence of random variables that take non-negative integer values.

To achieve this, it suffices to show that $\mathbb{E}[Y_i] \leq \frac{63}{64}\mathbb{E}[Y_{i-1}]$ for every $i \in \mathbb{N}$. We start by establishing that $\mathbb{E}[Y_i \mid Y_{i-1}] \leq \frac{63}{64}Y_{i-1}$ for each $i \in \mathbb{N}$.

Consider the case when $Y_{i-1} \geq 1$, i.e., $\mathsf{cc}(G_{i-1}) \geq 2$. Applying Lemma 3.1, we have:

$$\mathbb{E}[Y_i \mid Y_{i-1}] = \mathbb{E}[\mathsf{cc}(G_i) \mid \mathsf{cc}(G_{i-1}) = Y_{i-1} + 1] - 1 \leq \frac{63}{64}(Y_{i-1} + 1) - 1 \leq \frac{63}{64}Y_{i-1}.$$

On the other hand, if i is such that $Y_{i-1} = 0$, then $Y_i = 0$. Thus, $\mathbb{E}[Y_i \mid Y_{i-1}] \leq \frac{63}{64} Y_{i-1}$ holds trivially. Hence, we can conclude:

$$\mathbb{E}[Y_i] = \mathbb{E}[\mathbb{E}[Y_i \mid Y_{i-1}]] \leq \mathbb{E}\left[\frac{63}{64} Y_{i-1}\right] = \frac{63}{64}\mathbb{E}[Y_{i-1}] \quad \text{, for all } i \in \mathbb{N}.$$

This implies:

$$\mathbb{E}[Y_t] \leq \left(\frac{63}{64}\right)^t \mathbb{E}[Y_0] = \left(\frac{63}{64}\right)^t \cdot (n-1).$$

Since $t = c \log n$, we choose c to be sufficiently large such that $\mathbb{E}[Y_t] \leq \frac{1}{\text{poly}(n)}$. Applying Markov's inequality, we have $\Pr[Y_t \geq 1] \leq \frac{1}{\text{poly}(n)}$. Since Y_t only takes non-negative integer values, it follows that $Y_t = 0$ holds w.h.p. Therefore, we conclude that the protocol terminates in $O(\log n)$ phases w.h.p. ◀

Now we are ready to prove Theorem 1.1.

▶ **Theorem 1.1.** *There is a protocol in the* GOSSIP-reply (b) *model that can construct a star overlay in* $O\left(\log n \cdot \max\left(\frac{\log n}{b}, 1\right)\right)$ *rounds with* $O\left(n \log n \cdot \max\left(\frac{\log n}{b}, 1\right)\right)$ *messages w.h.p.*

Proof. The correctness of this algorithm is obvious since a leader is maintained and known to all nodes in the clusters after each phase. Once the algorithm terminates, there will be only one cluster and all nodes will agree on a single leader. Therefore, at the end of the algorithm, we have constructed a star overlay network.

By Lemma 3.2, we know that the algorithm terminates in $O(\log n)$ phases w.h.p. Now we only need to check that it takes $O(1)$ rounds in each phase in the GOSSIP-reply $(\log n)$ model to conclude that the algorithm terminates in $O(\log n)$ rounds with high probability. To be exact, we need seven rounds (described from the point of view of a cluster leader): 4 rounds to run FindOutgoing; 1 round to send the *requesting message* and receive the new leader; 1 round to resend the *requesting message* to the cluster leaders and receive an *accepting message* or a *rejecting message*; and 1 round to distribute the new leader identifier to the old cluster members. During this process, every cluster member just keeps sending requests to their old leader for the identifier of the new leader.

For GOSSIP-reply(b), where $b \in O(\log n)$, we can simulate one round of the above process with $O\left(\frac{\log n}{b}\right)$ rounds and arrive at our conclusion. The $O\left(n \log n \cdot \max\left(\frac{\log n}{b}, 1\right)\right)$ messages w.h.p. total message complexity follows from the restriction of the model that at most $O(n)$ messages are sent in each round. ◀

4 Well-formed tree overlay construction in the HYBRID model

In this section, we describe a protocol for well-formed tree (WFT) construction in the HYBRID$(\log n, 1, \log n)$ model. Due to Lemma 1.5 adapted from results in [26, 19], we can convert a well-formed tree overlay to a constant-degree expander overlay via only global communications in the HYBRID$(\log n, 1, \log n)$ model with an additive $O(\log^2 n)$ round complexity and additive $O(\log^2 n)$ messages for each node. Therefore, we will focus on describing the WFT construction protocol.

4.1 Algorithm

Now we describe our protocol HybridWFT to build the well-formed tree to prove Theorem 1.4. The overall structure of the algorithm is similar to that in Section 3. At the start, each node is a cluster, i.e., G_0 consists only of isolated nodes. The algorithm proceeds in phases where in each phase a constant fraction of the clusters are merged in expectation. At the end of $O(\log n)$ phases, there will be only one cluster (per connected component of the input graph) w.h.p. We maintain the following invariant in each cluster after each phase: the subgraph induced by each cluster is a satisfactory tree ($O(\log n)$-degree, $O(\log n)$-depth) that spans the cluster. More specifically, each node in the cluster knows its parent and children, as well as the root of the satisfactory tree.

In each phase, there will be 3 major steps. Unless mentioned otherwise, all communications are done over the global channel.

Sampling step. This step aims to sample an outgoing edge from each cluster. We run the protocol FindOutgoing(r) from the root r and find an outgoing edge (from the cluster) with constant probability. This takes $O(D(T_x)) = O(\log |T_x|)$ rounds and $O(|T_x|)$ messages. Next, the root node samples a random color $\chi \in \{\mathsf{Red}, \mathsf{Blue}\}$. Finally, r broadcasts the selected outgoing edge and the color in the cluster.

Grouping step. The selected node in each cluster will send a merging request along the selected outgoing edge. The merging request contains the color and the root identifier of the requesting cluster. Since we sample from the local edges, all merging requests will be over the local edges. Then each node receiving a request accepts the request if and only if the accepting cluster is Red and the requesting cluster is Blue. The accepting node sends an accepting message with the identifier of the accepting node via the local edge.

To improve the probability of low local communication, we will need to reduce the effective neighbors of Blue nodes. We do this via the following procedure. Each node v receiving a request will compute a matching over its rejected neighbors. Then v will send the *rejecting message* along with its matched *regrouping cluster* to each rejected requesting node. Then each rejected node will send a *regrouping message* to its matched *regrouping cluster* over the global network. The matched pairs will exchange their cluster leader identifier to decide on a new leader based on who has a larger identifier.

▶ **Lemma 4.1.** *The cluster graph $H^C = (\mathsf{CC}(G_i), E^C)$ is a forest, where $A, B \in \mathsf{CC}(G_i)$ are adjacent in H^C if and only if one accepts the other or one is matched with the other. Moreover, each tree in H^C has a diameter at most* 3.

Proof. First, we call the edges from the accepted request E_a^C and the edges from the matching E_m^C. Then $E^C = E_a^C \cup E_m^C$. Since each cluster only initiates one request and we only accept requests from Blue to Red, E_a^C induces a forest. Otherwise, there will be a cycle which consists of clusters of alternating colors. Then each cluster will be accepting some requests since each cluster only sends one request. However, this is impossible due to our acceptance rule because each accepting cluster (Red) will not be accepted, and each accepted cluster (Blue) will not be accepting any cluster. Since E_m^C comes from matching the rejected nodes, and all rejected clusters are not connected by accepting edges, H^C is a forest.

For the diameter, see that the connected components induced by E_a^C are stars centered at a Red cluster. This is because each Blue cluster rejects all requests and each request from a Red cluster is rejected. Then, we perform a case analysis for the components connected by E_m^C. Let $\{A, B\} \in E_m^C$. If A, B are both rejected Blue clusters, then this component has diameter 1. If A, B are both rejected Red clusters, then this component has a diameter at most 3. If A is Red and B is Blue, this component has diameter at most 2. ◀

Define $H = (V, E(G_i) \cup E_a \cup E_m)$, where E_a are edges from the *accepting messages* and E_m are edges from the *regrouping messages*. We call each connected component in H a grouped cluster in phase i. Note that each grouped cluster has agreed on a unique leader. We will now perform the merging step to transform each grouped cluster into a satisfactory tree.

To illustrate the grouping step clearly, we show a possible execution of the grouping step in Figure 1.

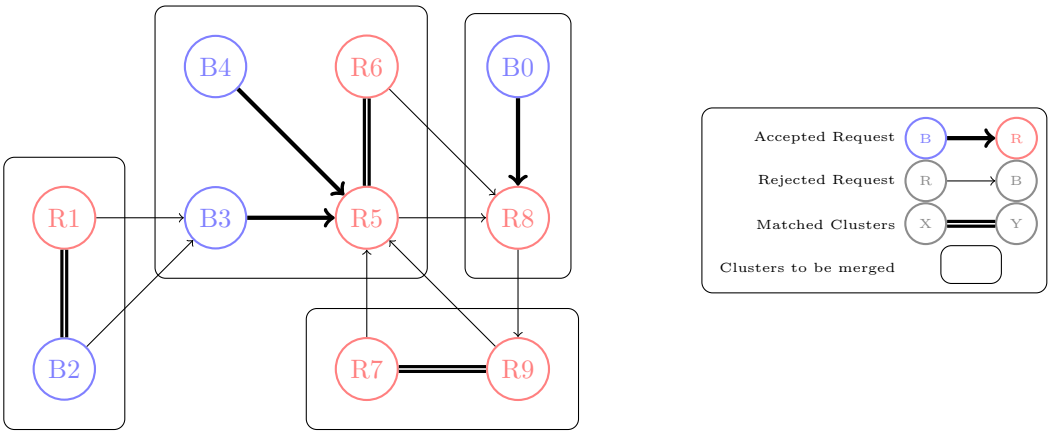


Figure 1 A possible grouping step.

Merging step. Each node in a cluster that acts on behalf of the leader (to reply to requests) will inform its leader that it is the new leader of a grouped cluster. Each new leader will initiate a re-rooting process via a breadth-first search style broadcast, where each node will change its parent and children according to their distance from the new leader. Those requesting nodes that received an accepting message will re-root the requesting cluster towards the new leader by relaying this broadcast in the requesting cluster.

Let v be the new leader of the grouped cluster. We now have a tree T_v rooted at v after re-rooting. T_v has depth $O(\log n)$, since each original cluster has depth $O(\log n)$ and H^C has diameter at most 3 due to Lemma 4.1. However, this tree might have a maximum degree up to $O(\Delta)$, due to the accepting edges i.e., edges in E_a. Therefore, we will perform the following transformation similar to the merging step in [24] to maintain the invariant. However, we made some crucial adaptation to the pointer jumping step to reduce message complexity at the cost of introducing randomness. First, we transform the tree into a child-sibling tree. Each node v will sort its children in some arbitrary order and then attach itself at the head of this order. Then for each child u in this order, v will send the previous and next node in the order. The last node will receive no next node. In this way, each node keeps at most one child and one sibling. By viewing the sibling as a child, we have constructed a binary tree. Then, we can proceed with the same Euler Tour technique to turn this into a cycle of virtual nodes. Finally, we perform a procedure RC2T (described in the full version [14]) to turn this cycle of virtual nodes into a tree with $O(\log n)$ degrees and $O(\log n)$ depth.

The above steps to construct a satisfactory tree after re-rooting are described in more detail in the full version [14], where we show that running this process for $O(\log n)$ times can be done in $O(\log^2 n)$ rounds with $O(n \log n)$ messages. Moreover, each node v uses at most $O(\deg_G(v) + \log n)$ messages.

Post-processing. After all the phases terminates, we get a cluster with a satisfactory tree overlay. We can now run one iteration of the deterministic well-formed tree construction in the full version to turn this satisfactory tree to a well-formed tree with additive $O(\log n)$ rounds and $O(n \log n)$ messages.

The formal analysis of the protocol HybridWFT is described in the full version [14], hence proving Theorem 1.4.

▶ **Theorem 1.4.** *There is a protocol in the* HYBRID $(\log n, 1, \log n)$ *model that can construct a well-formed tree overlay from any input graph G in $O\left(\log^2 n\right)$ rounds with $O\left(n \log n\right)$ messages w.h.p. Moreover, each node v sends and receives at most $O(\deg_G(v) + \log n)$ messages throughout the protocol.*

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.

2 Emmanuelle Anceaume, Maria Gradinariu, and Aina Ravoaja. Incentives for p2p fair resource sharing. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 253–260. IEEE, 2005. doi:10.1109/P2P.2005.17.

3 Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 145–154, 2005. doi:10.1145/1073970.1073991.

4 James Aspnes and Gauri Shah. Skip graphs. In *Proc. of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393. SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644170.

5 John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 69–79, 2019. doi:10.1145/3323165.3323195.

6 John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1280–1299. SIAM, 2020. doi:10.1137/1.9781611975994.78.

7 John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Storage and search in dynamic peer-to-peer networks. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 53–62, 2013. doi:10.1145/2486159.2486170.

8 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 74–83, 2013. doi:10.1145/2484239.2484275.

9 John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine leader election in dynamic networks. In *29th International Symposium on Distributed Computing (DISC)*, volume 9363 of *Lecture Notes in Computer Science*, pages 276–291, 2015. doi:10.1007/978-3-662-48653-5_19.

10 John Augustine, Gopal Pandurangan, Peter Robinson, Scott Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 350–369. IEEE, 2015. doi:10.1109/FOCS.2015.29.

11 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the Twenty-*

third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 551–569, 2012. doi:10.1137/1.9781611973099.47.

12 John Augustine and Sumathi Sivasubramaniam. Spartan: A framework for sparse robust addressable networks. In *Proc. of the 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1060–1069. IEEE, 2018. doi:10.1109/IPDPS.2018.00115.

13 Andrew Berns, Sukumar Ghosh, and Sriram V. Pemmaraju. Building self-stabilizing overlay networks with the transitive closure framework. *Theoretical Computer Science*, 512:2–14, 2013. doi:10.1016/J.TCS.2013.02.021.

14 Yi-Jun Chang, Yanyu Chen, and Gopinath Mishra. Overlay network construction: Improved overall and node-wise message complexity. *CoRR*, abs/2412.04771, 2024. doi:10.48550/arXiv.2412.04771.

15 Yi-Jun Chang, Oren Hecht, Dean Leitersdorf, and Philipp Schneider. Universally optimal information dissemination and shortest paths in the hybrid distributed model. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 380–390, 2024. doi:10.1145/3662158.3662791.

16 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

17 Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Sampling regular graphs and a peer-to-peer network. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–988. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070574.

18 Maximilian Drees, Robert Gmyr, and Christian Scheideler. Churn- and dos-resistant overlay networks based on network reconfiguration. In *Proc. of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 417–427. ACM, 2016. doi:10.1145/2935764.2935783.

19 Fabien Dufoulon, Michael Moorman, William K Moses Jr, and Gopal Pandurangan. Time-and communication-efficient overlay network construction via gossip. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ITCS.2024.42.

20 Michael Feldmann, Christian Scheideler, and Stefan Schmid. Survey on algorithms for self-stabilizing overlay networks. *ACM Computing Surveys*, 53(4):1–34, 2020.

21 Hillel Gazit. An optimal randomized parallel algorithm for finding connected components in a graph. *SIAM Journal on Computing*, 20(6):1046–1067, 1991. doi:10.1137/0220066.

22 Seth Gilbert, Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dconstructor: Efficient and robust network construction with polylogarithmic overhead. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 438–447. ACM, 2020. doi:10.1145/3382734.3405716.

23 C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

24 Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.137.

25 Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In *International Colloquium on Structural Information and Communication Complexity*, pages 262–276. Springer, 2019. doi:10.1007/978-3-030-24922-9_18.

26 Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. *Distributed Computing*, 36(3):313–347, 2023. doi:10.1007/S00446-023-00442-4.

27 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

28 Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip+: A self-stabilizing skip graph. *Journal of the ACM*, 61(6):36:1–36:26, 2014. doi:10.1145/2629695.

29 Tim Jacobs and Gopal Pandurangan. Stochastic analysis of a churn-tolerant structured peer-to-peer scheme. *Peer-to-Peer Networking and Applications*, 6(1):1–14, 2013. doi:10.1007/S12083-012-0124-Z.

30 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, Athens, Greece, 2011. ACM. doi:10.1145/1989284.1989289.

31 Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 71–80, 2015. doi:10.1145/2767386.2767405.

32 Fabian Kuhn and Philipp Schneider. Computing shortest paths and diameter in the hybrid network model. In *Proc. of the 39th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 109–118. ACM, 2020. doi:10.1145/3382734.3405719.

33 Vahid Heidaripour Lakhani, Leander Jehl, Rinke Hendriksen, and Vero Estrada-Galiñanes. Fair incentivization of bandwidth sharing in decentralized storage networks. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 39–44. IEEE, 2022. doi:10.1109/ICDCSW56584.2022.00017.

34 C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *IEEE INFOCOM*, pages 2133–2143, 2003.

35 Peter Mahlmann and Christian Schindelhauer. Distributed random digraph transformations for peer-to-peer networks. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 308–317, 2006. doi:10.1145/1148109.1148162.

36 Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient peer-to-peer network design for blockchains. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 428–437, 2020. doi:10.1145/3382734.3405704.

37 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2nd edition, 2005. doi:10.1017/CBO9780511813603.

38 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*, pages 1844–1860. SIAM, 2019. doi:10.1137/1.9781611975482.111.

39 Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 492–499, 2001. doi:10.1109/SFCS.2001.959925.

40 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Fast distributed algorithms for connectivity and MST in large graphs. *ACM Transactions on Parallel Computing (TOPC)*, 5(1):1–22, 2018. doi:10.1145/3209689.

41 Peter Robinson. Being fast means being chatty: The local information cost of graph spanners. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2105–2120. SIAM, 2021. doi:10.1137/1.9781611976465.126.

42 Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350. Springer, 2001. doi:10.1007/3-540-45518-3_18.

43 Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 149–160. ACM, 2001. doi:10.1145/383059.383071.

44 Huacheng Yu. Tight distributed sketching lower bound for connectivity. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1856–1873. SIAM, 2021. doi:10.1137/1.9781611976465.111.

A Related works

Various studies have explored methods for transforming arbitrary connected graphs into specific target topologies, such as expanders and well-formed tree [3, 24, 25, 26, 19]. Angluin et al. [3] were among the first to address this problem, demonstrating that any connected graph G with n nodes and m edges can be converted into a binary search tree with depth $O(\log n)$. Their algorithm requires $O(\Delta + \log n)$ rounds and $O(n(\Delta + \log n))$ messages, where Δ is the maximum degree of any node in the initial graph. The model they used allows each node to send only one message per round to a neighbor, and the resulting binary tree can be further transformed into other desirable structures like expanders, butterflies, or hypercubes. If the nodes are capable of sending and receiving a $O(\Delta \log n)$ number of messages per round, i.e. in P2P-CONGEST model, there exists a deterministic algorithm that operates in $O(\log^2 n)$ rounds, as shown in [24]. Recently, this has been improved to $O(\log^{3/2} n)$ rounds with high probability, as demonstrated in [25] for graphs with Δ as polylogarithmic.

Gilbert et al. [22] developed a different approach by designing a distributed protocol that efficiently reconfigures any connected network into a desired topology – such as an expander, hypercube, or Chord – with high probability. Here a node can send messages to all their neighbors in a round, regardless of their degree, resulting in faster communication for higher-degree nodes. Their protocol operates in $O(\text{polylog}\, n)$ rounds, utilizing messages of size $O(\log n)$ bits per link per round and achieving a message complexity of $\tilde{\Theta}(m)$.

Götte et al. [26] later introduced an algorithm for constructing a well-formed tree – a rooted tree with constant degree and $O(\log n)$ diameter – from any connected graph. Their protocol first builds an $O(\log n)$-degree expander, which can be further refined into the desired tree structure. The algorithm is optimal in terms of time, completing in $O(\log n)$ rounds, which aligns with the theoretical lower bound of $\Omega(\log n)$ for constructing such topologies from arbitrary graphs [26]. However, the message complexity remains $\tilde{\Theta}(m)$, as nodes are required to send and receive $d \log n$ messages per round, where d is the initial maximum degree. The key innovation in their approach is the use of short random walks to systematically improve the conductance of the graph, ultimately leading to the formation of robust expander structures. Very recently, Dufoulon et al. [19] introduces GOSSIP-reply$(\log^2 n)$ model and showed that a constant-degree expander can be constructed starting from any initial topology by spending $O(\log^5 n)$ rounds and with message complexity $O(n \log^5 n)$. This algorithm in Dufoulon et al. is the first protocol that achieves message complexity independent of m. Note that our result on GOSSIP-reply model (i.e., Corollary 1.3) is a strict improvement over the result of Dufoulon et al. in terms of both round and message complexity.

The HYBRID(α, β, γ) model, initially introduced by [6] for studying shortest paths, was further explored by [26], who showed that in the HYBRID$(\log n, 1, \log^2 n)$ model, an arbitrary topology can be transformed into a well-formed tree within $O(\log n)$ rounds. The message complexity of their algorithm is $O(m + n \log^3 n)$. In contrast, our result in the HYBRID$(\log n, 1, \log n)$ model (Theorem 1.4) achieves communication efficiency, albeit in $O(\log^2 n)$ rounds.

Research on overlay construction extends well beyond simple foundational examples, mainly due to the inherently dynamic nature of real-world overlay networks, which are often impacted by churn and adversarial behaviors. This research can be categorized into two primary areas: self-stabilizing overlays and synchronous overlay construction algorithms. Self-stabilizing overlays, which locally detect and correct invalid configurations, are extensively surveyed by Feuilloley et al. [20]. However, many of these algorithms lack definitive communication complexity bounds and provide limited guarantees for achieving polylogarithmic rounds [13, 28]. On the other hand, synchronous overlay construction algorithms are designed to preserve the desired network topology despite the presence of randomized or adversarial disruptions, thereby ensuring efficient load balancing and generating unpredictable topologies under certain error conditions [10, 18, 12, 25]. A significant advancement in this area is made by Gilbert et al. [22], who demonstrated how fast overlay construction can be maintained even in the presence of adversarial churn, assuming the network stays connected and stable for an adequate duration. Additionally, Augustine et al. [3] investigated graph realization problems, focusing on rapidly constructing graphs with specific degree distributions; however, their approach assumes the initial network is arranged as a line, which simplifies the task. The complexity of overlay construction increases when nodes have restricted communication capabilities, prompting research into the Node-Capacitated Clique (NCC) model, where each node can send and receive $O(\log n)$ messages per round [5]. Within the NCC model, efficient algorithms have been developed for various local problems such as MIS, matching, coloring, BFS tree, and MST [5]. Notably, Robinson [41] established that constructing constant stretch spanners within this model necessitates polynomial time. Similar complexities are encountered in hybrid network models that blend global overlay communication with traditional frameworks like LOCAL and CONGEST, where extensive communication abilities enable solving complex problems like APSP and SSSP effectively, though often with considerable local communication overheads [6, 15, 32, 20].

B Tradeoffs between complexity measures

There is a tradeoff among round complexity, message complexity, and balanced communication: If each node in G is only allowed to send $O(1)$ messages per round, then any algorithm to build a constant-degree overlay H requires $\Omega(\Delta(G))$ rounds [3]. To achieve both low round complexity and low message complexity independent of the initial degree of the input graph, certain nodes may need to send many more messages than others. This is observed in our first protocol in the GOSSIP-reply model in Section 3. Although it achieves good round and message complexities, it suffers from high regional communication where some nodes may need to communicate up to $\Omega(n)$ messages.

Node-wise message complexity. Node-wise message complexity can be seen as a measure that quantifies the aforementioned imbalance. The study of node-wise complexity is further motivated by the pursuit of fairness in the P2P network context. An extensive body of work is devoted to designing fair mechanisms to encourage users to contribute to the P2P network [2, 33]. However, besides offering incentives, it is essential to guarantee that users will incur low and fair costs when they join the network. A node may be discouraged from joining the network if it may potentially be selected as a crucial node of the network and perform much more work than other participating nodes.

Communication capacity versus round complexity. Communication capacity refers to the number of messages sent per node per round. This parameter captures the congestion in real-world networks. Intuitively, algorithms designed with more stringent communication capacity can perform better in real-world networks under congestion. There has been a series of works on improving the round complexity of the overlay network construction problem with more stringent communication capacity. Specifically, Angluin et al. [3] asked if there is an $O(\log n)$-round algorithm with $O(\Delta)$ communication capacity. While Götte et al. [26] answered this question affirmatively for the case that the communication capacity is $\Theta\left(\Delta + \log^2 n\right)$, in this work we present another tradeoff with $O\left(\log^2 n\right)$ round complexity and $O(\Delta + \log n)$ communication capacity, see Theorem 1.4 and Corollary 1.6.

IPS Lower Bounds for Formulas and Sum of ROABPs

Prerona Chatterjee
School of Computer Sciences, NISER, Bhubaneswar, India

Utsab Ghosal
Chennai Mathematical Institute, India

Partha Mukhopadhyay
Chennai Mathematical Institute, India

Amit Sinhababu
Chennai Mathematical Institute, India

Abstract

We give new lower bounds for the fragments of the Ideal Proof System (IPS) introduced by Grochow and Pitassi [23]. The Ideal Proof System is a central topic in algebraic proof complexity developed in the context of Nullstellensatz refutation [6] and simulates Extended Frege efficiently. Our main results are as follows.

- mult-$\mathsf{IPS}_{\mathsf{Lin}'}$: We prove nearly quadratic-size formula lower bound for multilinear refutation (over the Boolean hypercube) of a variant of the subset-sum axiom polynomial. Extending this, we obtain a nearly matching qualitative statement for a constant degree target polynomial.
- $\mathsf{IPS}_{\mathsf{Lin}'}$: Over the fields of characteristic zero, we prove exponential-size sum-of-ROABPs lower bound for the refutation of a variant of the subset-sum axiom polynomial. The result also extends over the fields of positive characteristics when the target polynomial is suitably modified. The modification is inspired by the recent results [24, 8].

The mult-$\mathsf{IPS}_{\mathsf{Lin}'}$ lower bound result is obtained by combining the quadratic-size formula lower bound technique of Kalorkoti [29] with some additional ideas. The proof technique of $\mathsf{IPS}_{\mathsf{Lin}'}$ lower bound result is inspired by the recent lower bound result of Chatterjee, Kush, Saraf and Shpilka [17].

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Proof complexity

Keywords and phrases Ideal Proof System, Lower Bound, Algebraic Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.22

Related Version *Full Version*: https://arxiv.org/abs/2507.09515 [15]

Funding *Utsab Ghosal*: Partially supported by Infosys grant.
Partha Mukhopadhyay: Partially supported by Infosys grant.
Amit Sinhababu: Partially supported by Infosys grant.

Acknowledgements We thank the anonymous reviewers for their valuable feedback and suggestions.

1 Introduction

The main goal in propositional proof complexity is to prove lower bounds for computational resources required to prove propositional tautologies. This task in its full generality is strongly related to the separation problem of NP and coNP as shown by Cook and Reckhow [18]. Among the propositional proof systems, the Frege proof system is very well studied since the early work by Reckhow [38]. In Frege proofs, the propositions are computable by *formulas* and lower bounds for the Frege system remain notoriously open. In the restricted setting, strong lower bounds for AC^0-Frege proof systems are known [1, 7, 30, 36, 26, 25]. In an


© Prerona Chatterjee, Utsab Ghosal, Partha Mukhopadhyay, and Amit Sinhababu;
licensed under Creative Commons License CC-BY 4.0

attempt to address the Frege lower bounds via algebraic methods, Beame, Cook, and Hoover [5], and Pitassi and Impagliazzo [35], introduced the Nullstellensatz proof system. The weak version of Hilbert's Nullstellensatz says that a set of polynomials (usually called *axioms*) $f_1(\bar{x}), \ldots, f_m(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ is unsatisfiable (over the algebraic closure of $\mathbb{F}$) if and only if there are polynomials $g_1(\bar{x}), \ldots, g_m(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ such that $\sum_{j=1}^{m} g_j(\bar{x}) f_j(\bar{x}) = 1$. The coefficients $g_1, \ldots, g_m$ are the Nullstellensatz refutations of the axioms. The *degree* and the *sparsity* are two important notions of the size measure for the refutations, and the lower bounds for them are known [6, 13, 37, 28, 12, 2]. However, the hard examples for such lower bound results usually admit polynomial-size Frege proofs. To overcome this, stronger algebraic proof systems were introduced that measure the size by the minimal size of the circuits computing the refutation polynomials $g_i(\bar{x})$. More precisely, this led to the Ideal Proof System (IPS) formulated by Grochow and Pitassi [23], where the refutation of $f_1(\bar{x}), \ldots, f_m(\bar{x})$ over the Boolean hypercube satisfies the equation

$$\sum_i g_i(\bar{x}) f_i(\bar{x}) + \sum_j h_j(\bar{x})(x_j^2 - x_j) = 1,$$

and $g_i(\bar{x}), h_j(\bar{x})$ are represented by algebraic circuits. The size of an IPS refutation is the total size of the circuits computing the polynomials g_i and h_i. Further formal description is given in Definition 2.7. The work of Grochow and Pitassi [23, 34] shows that IPS is powerful enough to polynomially simulate the Frege and the Extended Frege systems, making IPS lower bounds an important avenue of further research. The work of Forbes, Sphilka, Tzameret, and Wigderson [21] addresses fragments of linear IPS in models like ROABPs, multilinear formulas (both unrestricted and constant depth), and for some classes of constant depth formulas. In another interesting line of work, Lee, Tzameret, and Wang [32] connect the Frege lower bounds with noncommutative IPS lower bounds. More recently, the breakthrough result for the constant-depth circuit lower bounds by Limaye, Srinivasan, and Tavenas [33] led to a flurry of activities in this area [3, 22, 24]. Even over fields of positive characteristic, new IPS lower bounds were shown [8, 19]. These results were inspired by the result of Forbes [20] on constant-depth circuit lower bounds in positive characteristic. It might be worthwhile to note that none of these lower bounds are against polynomials that are arithmetizations of unsatisfiable CNFs. Therefore, it remains an important open problem to prove similar lower bounds for such polynomials.

In this paper, we address the IPS lower bound problem in the setting of general algebraic formulas. Note that the current best-known (general) formula size lower bound for an explicit polynomial is given by techniques due to Kalorkoti [29][1]. We lift this result to the IPS setting by giving nearly similar quality lower bounds for general formulas, and the axiom polynomial is a variant of the subset-sum polynomial. More precisely, we prove the lower bound for mult-IPS$_{\mathsf{Lin}'}$ as detailed in Subsection 1.1. In fact, we are able to show such a lower bound (slightly weaker) even for a subset-sum axiom polynomial of *constant degree*. In recent times, proving polynomial-quality lower bounds for constant-degree polynomials has received considerable attention since this is generally considered to be an avenue to improve the state-of-the-art polynomial-quality lower bounds for various algebraic models [27, 14].

Next, we consider the IPS lower bound question for the sum-of-ROABPs model. Besides being a natural extension to the works that study the same question for ROABPs [21, 24, 8, 19], this model has an additional motivation in the context of VBP vs VNP conjecture. In particular, inspired by the connection that Bhargav, Dwivedi, and Saxena observe in the

[1] Also see [16], [40, Section 3.2].

context of Valiant's VBP vs VNP conjecture [10], the work of Chatterjee, Kush, Saraf, and Shpilka [17] shows exponential-size lower bounds for sum-of-ROABPs model. Our result can be seen as a lift of those in [17] to the IPS setting.

Both the results use the functional lower bound technique from [21, Lemma 5.1]. In particular, it shows that proving any refutation of the unsatisfiable system $\{f = 0,\ \bar{x}^2 - \bar{x} = 0\}$ is not in a circuit class $\mathcal{C}$ is equivalent to proving that each $g(\bar{x})$ which agrees with $\frac{1}{f(\bar{x})}$ over the Boolean hypercube, is not in $\mathcal{C}$.

For our first result, to show that any multilinear-$\mathsf{IPS}_{\mathsf{Lin}'}$ refutation of the system $\{f = 0, \bar{x}^2 - \bar{x} = 0\}$ requires a quadratic-size formula lower bound, note that it suffices to show that the unique multilinear polynomial $g(\bar{x})$ which agrees with $\frac{1}{f(\bar{x})}$ over the Boolean hypercube also requires a quadratic-size formula lower bound. This follows from the properties of the multilinear-$\mathsf{IPS}_{\mathsf{Lin}'}$ model as explained in Definition 2.7, Definition 2.8. This was first formulated in the work of Govindasamy, Hakoniemi and Tzameret [22].

For our second result, observe that the sum of ROABPs can be multilinearized, just like ROABPs (Proposition 4.4). Hence, to show $\mathsf{IPS}_{\mathsf{Lin}'}$ lower bound in the sum of ROABPs model for any refutation of the unsatisfiable system $\{f = 0,\ \bar{x}^2 - \bar{x} = 0\}$, it is sufficient to a lower bound on the size of any sum of ROABPs computing the unique multilinear polynomial $g(\bar{x})$ which agrees with $\frac{1}{f(\bar{x})}$ over the Boolean hypercube.

1.1 Our Results

Formula Lower Bounds

We first define the target polynomial f for the unsatisfiable system.

Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$ be two sets of variables. We partition X into $N = \frac{n}{\log n}$ parts $\{X_1, \ldots, X_N\}$ where $X_i = \{x_{(i-1)\log n+1}, \ldots, x_{i\log n}\}$ for each $i \in N$. Clearly, the size of each X_i is $\log n$.

Fix any $i \in [N]$. For each subset $S \subseteq X_i$, let $\Pi_S \in \{0,1\}^{\log n}$ be its characteristic vector, and define $t(S) = \sum_{j=1}^{\log n} \Pi_S[j] \cdot 2^{j-1}$. Clearly, for any $s \in [0, \ldots, n-1]$ there is a unique S such that $s = t(S)$. We define the polynomials,

$$f'(\bar{x}, \bar{y}) = \sum_{i=1}^{N} \sum_{S \subseteq X_i} \left(\prod_{j \in S} x_j \cdot y_{t(S)} \right).$$

$$f(\bar{x}, \bar{y}) = \begin{cases} f' + 1 & \text{if the characteristic of } \ \mathbb{F} = 0 \\ f' + \beta & \text{if the characteristic of } \mathbb{F} = p > 0,\ k > 0 \in \mathbb{Z}, \\ & \beta \in \mathbb{F}_{p^{k+1}} \setminus \mathbb{F}_{p^k} \text{ and } f' \in \mathbb{F}_{p^k}[X, Y] \end{cases} \tag{1}$$

Clearly, $f = 0$ is unsatisfiable over the Boolean hypercube. Using this unsatisfiable system $\{f = 0, X^2 - X = 0, Y^2 - Y = 0\}$, we show our first lower bound result. Note that the degree of f is $\log n + 1$ and sparsity is $O\left(\frac{n^2}{\log n}\right)$.

▶ **Theorem 1.1.** *Consider the polynomial f is defined in Equation 1. Then the following statements are true :*

- *Over the fields of characteristic 0, any formula computing a* mult-$\mathsf{IPS}_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system $\{f = 0, X^2 - X = 0, Y^2 - Y = 0\}$ must have size at least $\Omega\left(\frac{n^2}{\log n}\right)$.*

- *Over the fields of characteristic p, any formula computing the* mult-IPS$_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system of equations* $\left\{f = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ *must have size at least* $\Omega\left(\frac{n^2}{\log n}\right)$.

Next, we describe a constant degree set-multilinear polynomial which is unsatisfiable over the Boolean hypercube such that any multilinear proof of unsatisfiability requires near quadratic formula size. We first state a fact about the set-multilinear polynomials.

▶ **Fact 1.2.** *Let* $X = \sqcup_{i=1}^{n} X_i$ *be a set of variables such that* $|X_i| = \ell$ *and* $X_i = \{x_{i,j} \ : \ j \in [\ell]\}$. *Then the number of set-multilinear monomials respecting the partition* $X_1, \ldots, X_n$ *is* ℓ^n. *Another way to see it is that any set-multilinear monomial* $m = \prod_{i=1}^{n} x_{i,j_i}$ *defines uniquely a map* $\tau_m : [n] \to [\ell]$, *with* $\tau_m(i) = j_i$. *The number of such possible distinct maps is* ℓ^n.

We now define the constant-degree polynomial.

Let $c > 3$. Further, let $X = \left\{x_{i,j} \ : \ i \in [c], j \in [n^2/c]\right\}$ and $Y = \{y_{i,j} \ : \ i, j \in [n]\}$ be sets of variables. The polynomial we will be defining, say $f \in \mathbb{F}[X, Y]$, is set-multilinear with respect to the partition $\{X_1, \ldots, X_c, Y\}$ where $X_i = \left\{x_{i,j} \ : \ j \in [n^2/c]\right\}$.

We further partition X_i into $n^{2(1-1/c)}/c$ parts, each of size $n^{2/c}$, and denote it by $X_{i,k}$ for $k \in [n^{2(1-1/c)}/c]$. So now, for any $i \in [c]$ and $k \in [n^{2(1-1/c)}/c]$,

$$X_{i,k} = \left\{x_{i,j} \ : \ j \in \left\{(k-1)(n^{2/c}) + 1, \ldots, k(n^{2/c})\right\}\right\}$$

By Fact 1.2, it is easy to see that the number of set-multilinear monomials over the variable set $X^{(k)} = X_{1,k} \sqcup \cdots \sqcup X_{c,k}$ is $(n^{2/c})^c = n^2$.

Note that $|Y| = n^2$. Thus, for any $k \in [n^{2(1-1/c)}/c]$, we can define a bijection, say $\pi_k : \mathcal{M}_{\mathrm{sm}}[X^{(k)}] \to Y$, which maps each set-multilinear monomial to a unique variable in Y. For any such k, we define the set-multilinear polynomial h_k of degree $c + 1$, over the variable set $X_{1,k} \sqcup \cdots \sqcup X_{c,k} \sqcup Y$, as $h_k = \sum_{m \in \mathcal{M}_{\mathrm{sm}}[X^{(k)} \sqcup Y]} m \cdot \pi_i(m)$. We then define the polynomial,

$$h'(X, Y) = \sum_{k=1}^{n^{2(1-1/c)}/c} h_k. \tag{2}$$

$$h(X, Y) = \begin{cases} h' + 1 & \text{if the characteristic of } \ \mathbb{F} = 0 \\ h' + \beta & \text{if the characteristic is } p > 0, \ k > 0 \in \mathbb{Z}, \\ & \beta \in \mathbb{F}_{p^{k+1}} \setminus \mathbb{F}_{p^k} \text{ and } h' \in \mathbb{F}_{p^k}[X, Y] \end{cases} \tag{3}$$

Note that the system of equations $\left\{h = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ is clearly unsatisfiable. Further h is a polynomial over n^2 variables of sparsity $n^{4-2/c}/c$ and degree $c + 1$. Using this system, we show our second lower bound result.

▶ **Theorem 1.3.** *Consider the polynomial h in Equation 3. Then the following statements are true.*

- *Over the fields of characteristic* 0, *any formula computing a* mult-IPS$_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system* $\left\{h = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ *must have size at least* $\Omega(n^{4-2/c})$.
- *Over the fields of characteristic p, any formula computing a* mult-IPS$_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system* $\left\{h = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ *must have size at-least* $\Omega\left(n^{4-2/c}\right)$.

▶ Remark 1.4. Since f and h are sparse polynomials, they can be obtained by substitutions of monomials in the usual subset-sum axiom polynomial of the form $\sum_i z_i - \gamma$. See the full version [15] for further details.

▶ Remark 1.5. We also note that the unsatisfiable systems given by f and h have *non-multilinear* refutations of polynomial size constant depth formulas. This follows easily from the known results [21, 8]. A proof sketch is given in the full version [15] for completeness.

Sum of ROABPs Lower Bounds

Over the fields of characteristic 0, we use a variant of the subset-sum polynomial to show an exponential-size lower bound for the $\mathsf{IPS}_{\mathsf{Lin}'}$ refutations in the sum of ROABPs model. The same polynomial has been used earlier to prove an exponential-size lower bound for $\mathsf{IPS}_{\mathsf{Lin}'}$ refutations in (any order) ROABPs model [21]. More precisely, we prove the following.

▶ **Theorem 1.6.** *For $X = \{x_0, \ldots, x_{2n-1}\}$, $T = \{t_{i,j} \ : \ i, j \in [0, \ldots, 2n-1]$ with $i < j\}$ and $\beta = 2\binom{2n}{2}$, let $f \in \mathbb{F}[X, T]$ be the polynomial defined as*

$$f = \left(\sum_{0 \le i < j \le 2n-1} t_{i,j} x_i x_j \right) - \beta$$

which is unsatisfiable over the Boolean hypercube. Then there exists $\gamma > 0$ such that the total width of any sum of ROABP *computing the linear proof of unsatisfiability ($\mathsf{IPS}_{\mathsf{Lin}'}$), for the system $\{f = 0, X^2 - X = 0, T^2 - T = 0\}$, is at least $\exp(n^\gamma)$.*

Similar lower bounds can be achieved over the fields of positive characteristic as well. We refer the readers to Subsection 4.3 for further details.

▶ Remark 1.7. Further, we note that there is a *non-multilinear* refutation of the unsatisfiable system given by f which is computable by poly(n) size ABP. More details can be found in the full version [15].

1.2 Proof Sketches

Formula lower bounds

The main ideas behind the $\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$ lower bounds in the formula model (Theorem 1.1, and Theorem 1.3) are based on the techniques from [29] and the functional lower bound method of [21]. The functional method shows that the proof of a $\mathcal{C}$-$\mathsf{IPS}_{\mathsf{Lin}'}$ lower bound for the unsatisfiable system $\left\{f = 0, x_1^2 - x_1 = 0, \ldots, x_n^2 - x_n = 0\right\}$, is equivalent to the fact that every polynomial $g(\bar{x})$ that agrees with $\frac{1}{f(\bar{x})}$ over the Boolean hypercube satisfies that $g \notin \mathcal{C}$. Since our goal is to show a formula lower bound in the $\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$ model, it suffices to show that the unique multilinear polynomial g that agrees with $\frac{1}{f}$ over the Boolean hypercube requires nearly quadratic size formulas.

Let $X = \{x_1, \ldots, x_n\}$ and $f \in \mathbb{F}[X]$. For a subset $S \subseteq X$, one can express the polynomial $f = \sum_{m \in \mathcal{M}[S]} m \cdot f_m$, where $\mathcal{M}[S]$ is the set of monomials defined on S and $f_m \in \mathbb{F}[X \setminus S]$. Let $\mathsf{coeff}_S(f) := \{f_m \ : \ m \in \mathcal{M}[S], \ f_m \neq 0\}$ and $\mathsf{alg\text{-}rank}_S(f)$ be the algebraic rank of $\mathsf{coeff}_S(f)$. The main result of [29] states that, if f is computable by a formula of size s, then for any partition $X_1 \sqcup X_2 \sqcup \ldots \sqcup X_t$, we have that $s \geq \Omega\left(\sum_{i=1}^{t} \mathsf{alg\text{-}rank}_{X_i}(f)\right)$.

It is a well-known fact that for any set of polynomials, the algebraic rank is lower bounded by the algebraic rank of their leading or trailing monomials. We would like to apply Kalorkoti's method on the unique multilinear polynomial g that agrees with $\frac{1}{f}$ over the Boolean hypercube. To do that, we need to study the relationship between the support structure of f and g.

Note that our polynomials have the property that for any two monomials, the support of one is not contained in the other. We show that for any polynomial f that satisfies this property, $\mathsf{supp}(f) \subseteq \mathsf{supp}(g)$ (Proposition 3.1). Furthermore, if a monomial m cannot be written as a multilinearized product of monomials from $\mathsf{supp}(f)$, then its coefficient in g is 0 (Lemma 3.2). Using these, it can be observed that for any $S \subseteq X$, the trailing monomials (under graded-lex ordering) of $\mathsf{coeff}_S(g)$ are equal to the trailing monomials of $\mathsf{coeff}_S(f)$.

Under the partition $X_1 \sqcup \cdots \sqcup X_N$ considered in the definition of Equation 1, since $\mathsf{coeff}_{X_i}(f)$ is a set of algebraically independent monomials, the trailing monomials in $\mathsf{coeff}_{X_i}(g)$ are also algebraically independent. The result now follows from [29].

Sum of ROABPs Lower Bounds

The $\mathsf{IPS}_{\mathsf{Lin}'}$ lower bound for the sum of ROABPs model combines techniques from the works of Forbes, Shpilka, Tzameret and Wigderson [21] and Chatterjee, Kush, Saraf, and Shpilka [17]. Similar to the formula setting, it is enough to show a lower bound against all polynomials g that agree with $1/f$ over the Boolean hypercube. Additionally, since the sum of ROABPs can be efficiently multilinearized (Proposition 4.4), it is enough to show a lower bound against the unique multilinear polynomial satisfying the above property.

The proof broadly consists of two parts. The first part is to establish a structural weakness of the sum of ROABPs. Roughly speaking, under a random partition of the variables, the rank of the partial derivative matrix of a polynomial computed efficiently by a sum of ROABPs is low with high probability (Lemma 4.7). The proof is developed using the ideas implicit in [17]. On the other hand, for any balanced partition, the derivative matrix corresponding to the unique multilinear polynomial g that agrees with $1/f$ (where f is defined in Theorem 1.6) has high rank ([21]). The lower bound follows by combining these two statements.

2 Preliminaries

We begin by stating the Chernoff Bound.

▶ **Theorem 2.1** (Chernoff Bound). *Let $X_1, \ldots, X_n$ be a set of 0-1 independent random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$ is the expected value of X. Then for any $\delta \in (0,1)$, $\Pr[X \leq (1-\delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{2}\right)$.*

2.1 Notations

For a set X, a partition is written as $X = X_1 \sqcup \cdots \sqcup X_k$. The set of natural numbers is $\mathbb{N}$. For $n \in \mathbb{N}$, the set $\{1, \ldots, n\}$ is denoted by $[n]$. The symmetric group over $\{1, 2, \ldots, n\}$ is S_n. For notational clarity, sometime $\bar{x}$ or X is used for the set of variables $\{x_1, \ldots, x_n\}$. The set of all possible monomials over X is denoted by $\mathcal{M}(X)$. The set of equations $\left\{x_i^2 - x_i = 0 \ : \ i \in [n]\right\}$ is sometime shorten as $X^2 - X$. For $\{a_1, \ldots, a_n\} \subseteq \mathbb{N}$, we denote by $\bar{x}^{\bar{a}}$, the monomial $\prod_{i \in [n]} x_i^{a_i}$. Given a monomial $\bar{x}^{\bar{a}}$, define $\mathsf{supp}(\bar{x}^{\bar{a}})$ to be the set $\{x_i \ : \ a_i \geq 1\}$. Given a monomial $\bar{x}^{\bar{a}}$, we define $\mathsf{mult}(\bar{x}^{\bar{a}})$ to be the multilinear version of the monomial. That is, $\mathsf{mult}(\bar{x}^{\bar{a}}) = \prod_{i=1}^{n} x_i^{\min\{a_i, 1\}}$. By linearity, we extend this to define $\mathsf{mult}(f)$ for any polynomial $f \in \mathbb{F}[X]$. For a subset $S \subseteq [n]$, we denote by $\mathbb{1}_S : [n] \to \{0,1\}$ the characteristic function for S. That is, for any $i \in [n]$, $\mathbb{1}_S(i) = 1 \iff i \in S$.

2.2 Definitions

▶ **Definition 2.2** (Algebraic Formulas). *An algebraic formula C is a directed tree with a unique output gate (root) of out-degree 0, and input gates of in-degree 0 (leaves) labeled by variables $x_1, \ldots, x_n$ or constants from $\mathbb{F}$. The internal gates are labeled by $+$ or $\times$. Each gate v computes a polynomial f_v defined recursively: if v is an input, then $f_v = \mathsf{label}(v) \in \{x_1, \ldots, x_n\} \cup \mathbb{F}$; if $v = u \ \mathsf{op} \ w$ for $\mathsf{op} \in \{+, \times\}$, then $f_v = f_u \ \mathsf{op} \ f_w$. The polynomial computed at the output gate is the polynomial computed by the formula.*

▶ **Definition 2.3** (ROABPs). *A Read-once Oblivious ABP* (ROABP) *is a directed acyclic graph where the vertex set is partitioned into layers $0, 1, \ldots, n$ with directed edges only between adjacent layers (i to $i+1$). Layers 0 and n have a single vertex each (called the source s and terminal t respectively), whereas the other layers can have any number of vertices.*

The labels on the edges satisfy the property that for every $i \in [n]$, there is a unique $j \in [n]$ such that every edge in between layer $j-1$ and j is labelled by a univariate polynomial in x_i. The polynomial computed by any s-to-t path is the product of the edge labels on it and the polynomial computed by the ROABP *is the sum of all polynomials computed by such paths.*

An ROABP *is said to have order $\sigma \in S_n$ if for every $j \in [n]$, the edges in between layers $j-1$ and j are labelled by univariates in $x_{\sigma(j)}$. An* ROABP *is said to be multilinear if each of the edge labels are linear polynomials.*

The size of an ROABP *is the total number of vertices in it; the width of any layer in the* ROABP *is the number of vertices in it and the width of an* ROABP *is the width of its widest layer. A sum of* ROABP*s is defined as the sum of individual* ROABP*s, with total width equal to the sum of their widths.*

▶ **Definition 2.4** (Set-Multilinear Polynomial). *Let $(X_1, \ldots, X_d)$ be a partition of the variable set X, with $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\}$. A polynomial $f \in \mathbb{F}[X]$ is said to be set-multilinear with respect to the given partition if each monomial in f is of the form $(x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d})$.*

It is sometimes useful to think of the variables as coming from a matrix $M_{d \times n}$ where the i^{th} row is $\{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\}$ and a set-multilinear polynomial is one in which each monomial is constructed by picking exactly one variable from each row.

▶ **Definition 2.5** (Monomial Ordering). *Given a set of variables $X = \{x_1, \ldots, x_n\}$, let $(x_{i_1}, \ldots, x_{i_n})$ be a total ordering on it. We extend it to a total ordering $\succ$ on $\mathcal{M}(X)$ as follows. For any two distinct monomials $\bar{x}^{\bar{a}}, \bar{x}^{\bar{b}}$,*

- *if $\deg(\bar{x}^{\bar{a}}) > \deg(\bar{x}^{\bar{b}})$ then $\bar{x}^{\bar{a}} \succ \bar{x}^{\bar{b}}$;*
- *if $\deg(\bar{x}^{\bar{a}}) = \deg(\bar{x}^{\bar{b}})$, $a_{i_j} = b_{i_j}$ for every $j < j_0$ and $a_{i_{j_0}} > b_{i_{j_0}}$, then $\bar{x}^{\bar{a}} \succ \bar{x}^{\bar{b}}$.*

For a polynomial $f \in \mathbb{F}[X]$ and a total order $\succ$ on $\mathcal{M}[X]$, we denote the leading monomial *of f (largest monomial under $\succ$ that present in f) and* trailing monomial *of f (smallest monomial under $\succ$ that is present in f) by $\mathrm{LM}(f)$ and $\mathrm{TM}(f)$ respectively.*

For a subset $S \subseteq \mathbb{F}[X]$, we define $\mathrm{LM}(S) := \{\mathrm{LM}(f) : f \in S\}$. $\mathrm{TM}(S)$ is defined similarly.

▶ **Definition 2.6** (Partial Derivative Matrix [40, 39]). *Let $f \in \mathbb{F}[X]$ be a polynomial and (Y, Z) be a partition of X (that is, $X = Y \sqcup Z$). The partial derivative matrix of f with respect to (Y, Z), say $M_{Y,Z}(f)$, is defined as follows.*

- *The rows of $M_{Y,Z}(f)$ are indexed by monomials in the variables Y and the columns are indexed by monomials in the variables Z.*
- *Given monomials $m_Y = \bar{y}^{\bar{a}}$ and $m_Z = \bar{z}^{\bar{b}}$, the entry of $M_{Y,Z}(f)$ in the row labelled m_Y and column labelled m_z is the coefficient of $m_Y \cdot m_Z$ in f (denoted by $\mathsf{coeff}_{m_Y \cdot m_Z}(f)$).*

Ideal Proof System

We begin with the definition of the Ideal Proof System (IPS) and some of its restrictions.

▶ **Definition 2.7** (Ideal Proof System [21, 23]). *Let $f_1, \ldots, f_m \in \mathbb{F}[X]$ be a set of polynomials such that $\{f_1, \ldots, f_m, x_1^2 - x_1, \ldots, x_n^2 - x_n\}$ has no common solution over the Boolean hypercube*[2]*.*

A proof *of the unsatisfiability of this set of polynomial equations, in the* Ideal Proof System (IPS)*, is a polynomial $P(X, y_1, \ldots, y_m, z_1, \ldots, z_n) \in \mathbb{F}[X, Y, Z]$ such that the following holds:*

- $P(X, \bar{0}, \bar{0}) = 0$*;*
- $P(X, f_1, \ldots, f_m, x_1^2 - x_1, \ldots, x_n^2 - x_n) = 1$.

▶ **Definition 2.8** (Restrictions of IPS [22, 24]). *Some restrictions of the Ideal Proof System that we will be considering are as follows.*

- $\mathsf{IPS}_{\mathsf{Lin}}$*: A proof, P, in the Ideal Proof System is said to be in* $\mathsf{IPS}_{\mathsf{Lin}}$ *if it additionally satisfies the conditions $\forall i \in [m],\ j \in [n],\ \deg_{y_i}(P), \deg_{z_j}(P) \leq 1$.*
- $\mathsf{IPS}_{\mathsf{Lin}'}$*: A proof, P, in the Ideal Proof System is said to be in* $\mathsf{IPS}_{\mathsf{Lin}'}$ *if it only satisfies the conditions $\forall i \in [m], \deg_{y_i}(P) \leq 1$.*
- $\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$*: A proof, P, in* $\mathsf{IPS}_{\mathsf{Lin}'}$ *is said to be in* $\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$ *if it additionally satisfies the condition that $P(X, Y, \bar{0})$ is multilinear polynomial. Note that $P(X, \bar{0}, Z)$ is not necessarily multilinear.*
- $\mathcal{C}$-$\mathsf{IPS}_{\mathsf{Lin}'}$ *and* $\mathcal{C}$-$\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$*: For any polynomial class $\mathcal{C}$, a proof in* $\mathsf{IPS}_{\mathsf{Lin}'}$ *is said to be in* $\mathcal{C}$-$\mathsf{IPS}_{\mathsf{Lin}'}$ *if it is additionally contained in $\mathcal{C}$.* $\mathcal{C}$-$\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$ *is defined analogously.*

Algebraic Independence and Algebraic Rank

▶ **Definition 2.9.** *A set of polynomials $\{f_1, \ldots, f_m\} \subseteq \mathbb{F}[X]$ is said to be algebraically dependent over $\mathbb{F}$ if there exists a non-zero polynomial $A(y_1, \ldots, y_m) \in \mathbb{F}[Y]$ such that $A(f_1, \ldots, f_m) = 0$. If no such A exists, then $f_1, \ldots, f_m$ are said to be algebraically independent.*

Given any set of polynomials $S \subseteq \mathbb{F}[X]$, the algebraic rank of S is the size of largest algebraic independent subset of S.

▶ **Definition 2.10.** *Given a polynomial $f \in \mathbb{F}[X]$ and $S \subseteq X$, let $\mathcal{M}[S]$ be the set of all monomials that can be defined over the variables in S. Furthermore, for each $m \in \mathcal{M}[S]$, let $f_m \in \mathbb{F}[X \setminus S]$ be the unique polynomial such that $f = \sum_{m \in \mathcal{M}[S]} m \cdot f_m$. We define* $\mathsf{alg\text{-}rank}_S(f)$ *as the algebraic rank of the set* $\mathsf{coeff}_S(f) := \{f_m \ :\ m \in \mathcal{M}[S],\ f_m \neq 0\}$.

Here is a standard theorem on algebraic independence, that we need.

▶ **Theorem 2.11** ([31]). *Let $f_1, \ldots, f_k \in \mathbb{F}[X]$ and $\succ$ be a total ordering over the monomials. If $f_1, \ldots, f_k$ are algebraically dependent, then both sets $\{\mathrm{LM}(f_i) \mid i \in [1, 2, \ldots, k]\}$ and $\{\mathrm{TM}(f_i) \mid i \in [1, 2, \ldots, k]\}$ are algebraically dependent.*

2.3 Functional Lower Bound Method for proving lower bounds for IPS Proofs

We will be crucially using the following technique described in [21] for proving lower bounds against $\mathsf{IPS}_{\mathsf{Lin}}$ and $\mathsf{IPS}_{\mathsf{Lin}'}$.

---

[2] That is, there does not exist $\bar{x} \in \{0,1\}^n$ such that for every $i \in [n]$, $f_i(\bar{x}) = 0$.

▶ **Theorem 2.12** ([21, Lemma 5.1]). *Let $f \in \mathbb{F}[X]$ be a polynomial such that for some $\beta \neq 0 \in \mathbb{F}$ the system $\{f - \beta, X^2 - X\}$ has no common solutions over the Boolean hypercube. Further, let $\mathcal{C} \subseteq \mathbb{F}[X]$ be a class of polynomials that is closed under partial $\mathbb{F}$-assignments[3].*

If there does not exist any $g \in \mathcal{C}$ that satisfies $g(\bar{x}) = \frac{1}{f(\bar{x})-\beta}$ for every $\bar{x} \in \{0,1\}^n$, then there is no proof of unsatisfiability for the system $\{f - \beta, X^2 - X\}$ that is contained in $\mathcal{C}$-IPS$_{\mathsf{Lin}}$, $\mathcal{C}$-IPS$_{\mathsf{Lin}'}$.

This is called the functional lower bound method, since one needs to prove a lower bound against all polynomials which evaluate to the same value as $\frac{1}{f-\beta}$ over the entire Boolean hypercube.

▶ **Proposition 2.13.** *Let $f \in \mathbb{F}[X]$ and $\{f = 0, X^2 - X = 0\}$ be a unsatisfiable system against which we want to show $\mathcal{C}$-mult-IPS$_{\mathsf{Lin}'}$ lower bound. Let $g(X)$ be the unique multilinear polynomial that agrees with $\frac{1}{f(X)}$ over the Boolean hypercube. If $g \notin \mathcal{C}$ then the system does not have $\mathcal{C}$-mult-IPS$_{\mathsf{Lin}'}$ refutation.*

Proof. Let $C(X, y, Z)$ be a $\mathcal{C}$-mult-IPS$_{\mathsf{Lin}'}$ refutation for the unsatisfiable system. Assume $g(X)$ be the unique multilinear polynomial such that $g(X) = \frac{1}{f(X)} \pmod{X^2 - X}$. Since, f is the only non-Boolean axiom and C is linear in y, by Definition 2.8, $C(X, y, Z) = g(X) \cdot y + C'(X, Z, y)$ and $C(X, f, X^2 - X) = g(X) \cdot f(X) + \sum_{i=1}^{n} C'_i(X)(x_i^2 - x_i) = 1$. So, $C(X, y, 0) = y \cdot g(X) \implies C(X, 1, 0) = g(X)$. This implies $g(X) \in \mathcal{C}$, but this contradicts the assumption : $g \notin \mathcal{C}$. ◀

3 Lower Bound Against Formulas for mult-IPS$_{\mathsf{Lin}'}$ Proofs

In this section, we prove Theorem 1.1 and Theorem 1.3.

Given an unsatisfiable system $\{f = 0, X^2 - X = 0\}$, to prove the formula complexity lower bound of the mult-IPS$_{\mathsf{Lin}'}$ refutation for f, it suffices to consider the unique multilinear polynomial $g(X)$ such that, $g(X) = \frac{1}{f(X)} \pmod{X^2 - X}$.

Towards that, we first note a few structural properties of the polynomial $g(X)$.

3.1 Some Structural Results

▶ **Proposition 3.1.** *Let $\mathbb{F}$ be any field and $f \in \mathbb{F}[X]$ be a multilinear polynomial such that the system of equations $\{f = 0, X^2 - X = 0\}$ is unsatisfiable over $\{0,1\}^n$. Additionally, suppose f has the property that for any two monomials $m_i, m_j \in f$ with non-empty support, $\mathsf{supp}(m_i) \nsubseteq \mathsf{supp}(m_j)$ and $\mathsf{supp}(m_j) \nsubseteq \mathsf{supp}(m_i)$. Let $g \in \mathbb{F}[X]$ be the unique multilinear polynomial such that $g(X) = \frac{1}{f(X)} \pmod{X^2 - X}$.*

Then, for any monomial m whose coefficient in f is non-zero, its coefficient in g is also non-zero[4].

Proof. Let $f(X) = f'(X) - \beta$ where $f(0) = -\beta$. Note that since f is unsatisfiable, $\beta \neq 0$. The unique multilinear polynomial $g(X) = \frac{1}{f(X)} \pmod{X^2 - X}$ is given by the following.

$$g(\bar{x}) = \sum_{T \subseteq [n]} g(\mathbb{1}_T) \prod_{i \in T} x_i \prod_{i \notin T} (1 - x_i).$$

[3] If $f \in \mathbb{F}[X]$ with $f \in \mathcal{C}$ and $Y \subseteq X$, then $f|_{Y=\bar{a}}(X \setminus Y) \in \mathcal{C}$ for any $\bar{a} \in \mathbb{F}^{|Y|}$.

[4] If f is not multilinear, then for any non-multilinear monomial m with non-zero coefficient in f, the coefficient of $\mathsf{mult}(m)$ in g is non-zero.

Let m be a monomial with non-zero coefficient α in f and $S = \mathsf{supp}(m)$. Setting the variables $x_i \notin S$ to 0, it is easy to see that the coefficient of m in g (denoted by c_m), is given by the expression $c_m = \sum_{A \subseteq S} g(\mathbb{1}_A)(-1)^{|S \setminus A|}$. Notice that in the expression c_m, the term in the summand is $g(\mathbb{1}_S) = \frac{1}{f(\mathbb{1}_S)} = \frac{1}{\alpha - \beta}$ when $A = S$ and it is $g(\mathbb{1}_\emptyset)(-1)^{|S|} = \frac{(-1)^{|S|}}{-\beta}$ when $A = \emptyset$. Moreover, from the monomial support property, for any $S' \subset S$, $g(\mathbb{1}_{S'}) = -\frac{1}{\beta}$. Thus,

$$\begin{aligned} c_m &= \sum_{A \subseteq S} g(\mathbb{1}_A)(-1)^{|S \setminus A|} = \frac{1}{\alpha - \beta} - \frac{1}{\beta} \sum_{A \subset S} (-1)^{|S \setminus A|} \\ &= \frac{1}{\alpha - \beta} - \frac{1}{\beta} \left(\sum_{A \subseteq S} (-1)^{|S \setminus A|} - 1 \right) = \frac{1}{\alpha - \beta} + \frac{1}{\beta}. \end{aligned} \tag{4}$$

In the above Equation 4, we use the standard fact that

$$\sum_{A \subseteq S} (-1)^{|S \setminus A|} = \sum_{i=0}^{|S|} \binom{|S|}{i} (-1)^{|S|-i} = 0. \qquad \blacktriangleleft$$

We record some further structural properties in the following lemma. The lemma shows that any monomial m which is not expressible by the multilinearization of any product of monomials from $\mathsf{supp}(f)$, has coefficient 0 in g.

▶ **Lemma 3.2.** *Let $\mathbb{F}$ be any field and $f \in \mathbb{F}[X]$ be a multilinear polynomial such that $\{f = 0, X^2 - X = 0\}$ is unsatisfiable. Let $g \in \mathbb{F}[X]$ be the unique multilinear polynomial such that $g(X) = \frac{1}{f(X)} \pmod{X^2 - X}$. Consider a monomial m such that $m \neq \mathsf{mult}(\prod_{m \in S} m)$ for any subset $S \subseteq \mathsf{supp}(f)$. Then the coefficient of m in g is 0.*

Proof. Consider a monomial m such that $m \neq \mathsf{mult}(\prod_{m \in S} m)$ for any subset $S \subseteq \mathsf{supp}(f)$. The idea is to decompose m as a product of monomials m_1 and m_2 with the following property : There is a set $S_1 \subseteq \mathsf{supp}(f)$, such that $m_1 = \mathsf{mult}(\prod_{m' \in S_1} m')$ and $S_2 = \mathsf{supp}(m_2) \notin \mathsf{supp}(f)$. Moreover for any subset $T_1 \subseteq S_1$ and a nonempty subset $T_2 \subseteq S_2$, $T_1 \cup T_2 \notin \mathsf{supp}(f)$. It is not hard to see that such a decomposition can be constructed in a greedy manner.

Next, we make a few simple observations. Clearly, $S_1 \cap S_2 = \emptyset$ due to multilinearity. Moreover, $g(\mathbb{1}_{S_2}) = 1/f(0)$. Furthermore, for any subset $T_1 \subseteq S_1$ and $T_2 \subseteq S_2$,

$$g(\mathbb{1}_{T_1 \cup T_2}) = \frac{1}{f(\mathbb{1}_{T_1 \cup T_2})} = \frac{1}{f(\mathbb{1}_{T_1})}.$$

Recall that,

$$g(\bar{x}) = \sum_{T \subseteq [n]} g(\mathbb{1}_T) \prod_{i \in T} x_i \prod_{i \notin T} (1 - x_i).$$

Hence,

$$\begin{aligned} c_m = \sum_{A \subseteq S_1 \cup S_2} g(\mathbb{1}_A)(-1)^{|S_1 \cup S_2 \setminus A|} &= \sum_{A_1 \subseteq S_1} (-1)^{|S_1 \setminus A_1|} \sum_{A_2 \subseteq S_2} g(\mathbb{1}_{A_1 \cup A_2})(-1)^{|S_2 \setminus A_2|} \\ &= \sum_{A_1 \subseteq S_1} (-1)^{|S_1 \setminus A_1|} \sum_{A_2 \subseteq S_2} g(\mathbb{1}_{A_1})(-1)^{|S_2 \setminus A_2|} \\ &= \sum_{A_1 \subseteq S_1} g(\mathbb{1}_{A_1})(-1)^{|S_1 \setminus A_1|} \sum_{A_2 \subseteq S_2} (-1)^{|S_2 \setminus A_2|} = 0. \end{aligned}$$

Here we have used the fact that $\sum_{A_2 \subseteq S_2} (-1)^{|S_2 \setminus A_2|} = 0$ and $g(\mathbb{1}_{A_1 \cup A_2}) = g(\mathbb{1}_{A_1})$. $\blacktriangleleft$

3.2 The Lower Bound

In this section, we prove a near-quadratic mult-IPS$_{\mathsf{Lin}'}$ size lower bound in the formula setting. We use the structural results developed in Subsection 3.1 along with the lower bound technique of [29].

▶ **Theorem 3.3** (Kalorkoti, [29]). *Let $f \in \mathbb{F}[X]$ be a polynomial computed by a size s formula. Let $(X_1, X_2, \ldots, X_t)$ be any partition of the variables set X. Then s is at-least $\Omega\left(\sum_{i=1}^{t} \mathsf{alg\text{-}rank}_{X_i}(f)\right)$.*

Next, we recall the axiom polynomial from Subsection 1.1, for which we show the lower bound result for IPS$_{\mathsf{Lin}'}$ refutations.

Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$ be two sets of variables. We partition X into $N = \frac{n}{\log n}$ parts, $\{X_1, \ldots, X_N\}$ where $X_i = \{x_{(i-1)\log n+1}, \ldots, x_{i \log n}\}$. Clearly, the size of each X_i is $\log n$.

For a subset $S \subseteq X_i$, let $\Pi_S \in \{0,1\}^{\log n}$ be its characteristic vector, and define $t(S) = \sum_{j=1}^{\log n} \Pi_S[j] \cdot 2^{j-1}$. Now we define the polynomial,

$$f'(\bar{x}, \bar{y}) = \sum_{i=1}^{N} \sum_{S \subseteq X_i} \left(\prod_{j \in S} x_j \cdot y_{t(S)} \right).$$

$$f(\bar{x}, \bar{y}) = \begin{cases} f' + 1 & \text{if the characteristic of } \mathbb{F} = 0 \\ f' + \beta & \text{if the characteristic of } \mathbb{F} = p > 0,\ k > 0 \in \mathbb{Z}, \\ & \beta \in \mathbb{F}_{p^{k+1}} \setminus \mathbb{F}_{p^k} \text{ and } f' \in \mathbb{F}_{p^k}[X, Y] \end{cases} \tag{1}$$

We now prove the first lower bound of this section: a $\log n$-degree unsatisfiable system whose mult-IPS$_{\mathsf{Lin}'}$ refutation by formulas requires near-quadratic size. We first recall the statement.

▶ **Theorem 1.1.** *Consider the polynomial f is defined in Equation 1. Then the following statements are true :*

- *Over the fields of characteristic 0, any formula computing a mult-IPS$_{\mathsf{Lin}'}$ refutation of the unsatisfiable system $\{f = 0, X^2 - X = 0, Y^2 - Y = 0\}$ must have size at least $\Omega\left(\frac{n^2}{\log n}\right)$.*
- *Over the fields of characteristic p, any formula computing the mult-IPS$_{\mathsf{Lin}'}$ refutation of the unsatisfiable system of equations $\{f = 0, X^2 - X = 0, Y^2 - Y = 0\}$ must have size at least $\Omega\left(\frac{n^2}{\log n}\right)$.*

Proof. Let $g(X, Y)$ be the unique multilinear polynomial such that $g(X, Y) = \frac{1}{f(X,Y)} \pmod{X^2 - X, Y^2 - Y}$. For any two monomials $m_1, m_2 \in \mathsf{supp}(f)$, $\mathsf{supp}(m_1) \nsubseteq \mathsf{supp}(m_2)$ and $\mathsf{supp}(m_2) \nsubseteq \mathsf{supp}(m_1)$. Hence Proposition 3.1 implies that, all monomials of f appear in g with non-zero coefficients. Moreover, Lemma 3.2 implies every other monomial in g with non-zero coefficient can only be the multilinearized product of monomials from $\mathsf{supp}(f)$. Consider the variable partition $X = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_N$ as defined above, and fix any X_i. Order monomials with $X \succ Y$, then extend as in Definition 2.5, using any order within X and Y. Under this ordering, $\mathrm{TM}(\mathsf{coeff}_{X_i}(g)) = \{y_0, \ldots, y_{n-1}\}$, which is algebraically independent. So using Theorem 2.11, the set $\mathsf{coeff}_{X_i}(g)$ is also algebraically independent. Let g be computable by a formula of size s. Then using Theorem 3.3, we conclude

$$s \geq \Omega\left(\sum_{i=1}^{N} \mathsf{alg\text{-}rank}_{X_i}(g) + \mathsf{alg\text{-}rank}_{Y}(g) \right) = \Omega\left(\frac{n^2}{\log n} \right).$$

Here we have used $N = \frac{n}{\log n}$. Using Proposition 2.13, any mult-IPS$_{\mathsf{Lin}'}$ refutation of the unsatisfiable system $\left\{f = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ computed by a formula needs size at-least $\Omega\left(\frac{n^2}{\log n}\right)$. Note that both Lemma 3.2 and Proposition 3.1 are characteristic independent statement. So, under the given partition $X_1 \sqcup X_2 \sqcup \cdots \sqcup X_N$, the set $\mathrm{TM}(\mathsf{coeff}_{X_i}(g)) = \{y_0, \ldots, y_{n-1}\}$ remains unchanged for any X_i. The algebraic independence of this set is independent of the characteristic of the field. So, the lower bound works over characteristic p as well. ◀

Next we show an example of a constant degree unsatisfiable system of equations such that any refutation in mult-IPS$_{\mathsf{Lin}'}$ computed by formula needs near quadratic size. For that, first we recall polynomial h' from Equation 2. Let $c > 3$. Further, let $X = \left\{x_{i,j} \ : \ i \in [c], j \in [n^2/c]\right\}$ and $Y = \{y_{i,j} \ : \ i, j \in [n]\}$ be sets of variables. The polynomial h is set multilinear with respect to the partition $\bigsqcup_{i=1}^{c}\left(\bigcup_{k=1}^{n^{2(1-1/c)}/c} X_{i,k}\right) \bigsqcup Y$ where for any $i \in [c]$ and $k \in [n^{2(1-1/c)}/c]$,

$$X_{i,k} = \left\{x_{i,j} \ : \ j \in \left\{(k-1)(n^{2/c}) + 1, \ldots, k(n^{2/c})\right\}\right\}$$

$$h'(X, Y) = \sum_{k=1}^{n^{2(1-1/c)}/c} h_k. \tag{2}$$

Where each h_k is set-multilinear over $\bigsqcup_{i=1}^{c} X_{i,k} \bigsqcup Y$ described in Equation 2. Let $\bigsqcup_{i=1}^{c} X_{i,k} = X^{(k)}$ and $\left(\sqcup_{k=1}^{n^{2(1-1/c)}/c} X^{(k)}\right) \sqcup Y$ be a partition of variables. We use this partition to prove the mult-IPS$_{\mathsf{Lin}'}$-lower bound against constant degree unsatisfiable systems. We first recall the polynomial.

$$h(X, Y) = \begin{cases} h' + 1 & \text{if the characteristic of } \mathbb{F} = 0 \\ h' + \beta & \text{if the characteristic is } p > 0, \ k > 0 \in \mathbb{Z}, \\ & \beta \in \mathbb{F}_{p^{k+1}} \setminus \mathbb{F}_{p^k} \text{ and } h' \in \mathbb{F}_{p^k}[X, Y] \end{cases} \tag{3}$$

Next, we recall the theorem statement.

▶ **Theorem 1.3.** *Consider the polynomial h in Equation 3. Then the following statements are true.*

- *Over the fields of characteristic 0, any formula computing a* mult-IPS$_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system $\left\{h = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ must have size at least $\Omega(n^{4-2/c})$.*
- *Over the fields of characteristic p, any formula computing a* mult-IPS$_{\mathsf{Lin}'}$ *refutation of the unsatisfiable system $\left\{h = 0, X^2 - X = 0, Y^2 - Y = 0\right\}$ must have size at-least $\Omega\left(n^{4-2/c}\right)$.*

Proof. Let $g(X, Y)$ be the multilinear polynomial such that $g(X, Y) = \frac{1}{h(X,Y)} \pmod{X^2 - X, Y^2 - Y}$. For any two monomials $m_1, m_2 \in \mathsf{supp}(h)$, $\mathsf{supp}(m_1) \nsubseteq \mathsf{supp}(m_2)$ and $\mathsf{supp}(m_2) \nsubseteq \mathsf{supp}(m_1)$. Hence Proposition 3.1 implies all monomials of h appear in g with a non-zero coefficient. Moreover, Lemma 3.2 implies every other monomial in g with non-zero coefficient can only be the multilinearized product of monomials from $\mathsf{supp}(h)$. Consider the partition of variables $\left(\bigsqcup_{k=1}^{n^{2(1-1/c)}/c} X^{(k)}\right) \bigsqcup Y$. Consider the following monomial ordering: Over the variables, $X \succ Y$ and extend it to monomials naturally (Definition 2.5). Within variable set X (and similarly Y), choose any order. Under this ordering, $\mathrm{TM}(\mathsf{coeff}_{X^{(k)}}(g)) =$

$\{Y_{i,j} : i,j \in [n]\}$ for every $X^{(k)}$ where $k \in [n^{2(1-1/c)}/c]$. All these variables are algebraically independent. So, using Theorem 2.11, $\mathsf{alg\text{-}rank}_{X^{(k)}}(g) \geq n^2$. If g has a formula of size s, then using Theorem 3.3,

$$s \geq \Omega\left(\sum_{i=1}^{n^{2(1-1/c)}/c} \mathsf{alg\text{-}rank}_{X^i}(g) + \mathsf{alg\text{-}rank}_Y(g)\right) \geq \Omega\left(n^2 \cdot n^{2(1-1/c)}/c + 1\right) \geq \Omega\left(n^{4-2/c}\right).$$

So, any formula computing g needs size at least $\Omega(n^{4-2/c})$. Hence, any $\mathsf{mult\text{-}IPS}_{\mathsf{Lin}'}$ refutation of $\{h = 0, X^2 - X = 0, Y^2 - Y = 0\}$ computed by a formula needs size at-least $\Omega(n^{4-2/c})$ by Proposition 2.13. Since both Lemma 3.2 and Proposition 3.1 is a characteristic independent statement, the set $\mathrm{TM}(\mathsf{coeff}_{X^{(k)}}(g))$ remains unchanged and algebraically independent. So, the proof follows when characteristic of the field is positive. ◀

4 Lower Bound Against Sum of ROABPs for $\mathsf{IPS}_{\mathsf{Lin}'}$ Proofs

We begin with an observation.

▶ **Observation 4.1.** *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a multilinear polynomial computed by an* ROABP *of size s. Then there is a multilinear* ROABP *of size at most s computing the same polynomial.*

Proof. Consider the ROABP, say $\mathcal{A}$, computing f. We get a multilinear ROABP computing f by simply removing non-multilinear terms from the label of every edge in $\mathcal{A}$. Since f is multilinear, the contribution of the non-multilinear terms in the edge labels anyway cancel out at the end and therefore this does not affect the computation of f. ◀

We next state a couple of theorems from the work of Forbes, Shpilka, Tzameret, and Wigderson [21] that we will require.

▶ **Lemma 4.2** ([21, Lemma 3.7])**.** *Let $X = \{x_1, \ldots, x_n\}$ and $f \in \mathbb{F}[X]$ be a polynomial computed by an* ROABP *of width r. Then $r \geq \max_{i \in [n]} \{\mathrm{rank}(M_{Y_i,Z_i}(f))\}$ where $Y_i = \{x_1, \ldots, x_i\}$ and $Z_i = X \setminus Y_i$.*

▶ **Lemma 4.3** ([21, Proposition 4.5])**.** *Let $f \in \mathbb{F}[X]$ be a polynomial with individual degree of each variable being at most d. Further, suppose that f is computable by an* ROABP *of width r in some order. Then* $\mathsf{mult}(f)$ *can be computed by an* ROABP *of size* $\mathrm{poly}(r, n, d)$ *and width r that has the same order.*

Further there exist polynomials $h_1, \cdots, h_n \in \mathbb{F}[X]$ such that

- *for every $i \in [n]$, h_i has individual degree upper bounded by d;*
- *for every $i \in [n]$, h_i can be computed by an* ROABP *of size* $\mathrm{poly}(r, n, d)$ *and width r;*
- $f(\bar{x}) = \mathsf{mult}(f) + \sum_{i=1}^{n} h_i(x_i^2 - x_i)$.

We now use Lemma 4.3 to show a similar statement for a sum of ROABPs as well.

▶ **Proposition 4.4** (Multilinearization of Sum of ROABPs)**.** *Let $f \in \mathbb{F}[X]$ be a polynomial with individual degree at most d such that it is computable by a sum of t* ROABP*s, say $A_1, \ldots, A_t$, each with width at most r and potentially a different variable ordering. Let σ_i be the variable ordering of A_i. Then,*

- $\mathsf{mult}(f)$ *is computable by a sum of t multilinear* ROABP*s $B_1, \ldots, B_t$, where each B_i has width at most r, size* $\mathrm{poly}(r, n, d)$ *and variable ordering σ_i. Further, if f_i was the polynomial computed by A_i, then the polynomial computed by B_i is* $\mathsf{mult}(f_i)$*;*

- *there exist polynomials $h_1, \ldots, h_n$, each of individual degree at most d and computable by a sum of t ROABPs of size $\mathrm{poly}(r, n, d)$ and width at most r, such that $f(\bar{x}) = \mathsf{mult}(f) + \sum_{i=1}^{n} h_i(x_i^2 - x_i)$.*

Proof. We use Lemma 4.3 to prove the statement. By the assumption of the lemma, f is computable by $\sum_{i=1}^{t} A_i$ where each A_i is an ROABP of width at most r and variable ordering σ_i. Let f_i be the polynomial computed by A_i. Note that the individual degree of f_i is at most d.

Fix any i arbitrarily. Using Lemma 4.3, $\mathsf{mult}(f_i)$ has an ROABP of width at most r, order σ_i and size $\mathrm{poly}(r, n, d)$. Moreover, there exist polynomials $h_{i,1}, \ldots, h_{i,n}$ of individual degree at most d such that $f_i = \mathsf{mult}(f_i) + \sum_{j=1}^{n} h_{i,j}(x_j^2 - x_j)$. Here, for each $j \in [n]$, $h_{i,j}$ can be computed by an ROABP of width at most r, order σ_i and size $\mathrm{poly}(r, n, d)$. Hence,

$$\begin{aligned} f = \sum_{i=1}^{t} f_i &= \sum_{i=1}^{t} \mathsf{mult}(f_i) + \sum_{i=1}^{t}\sum_{j=1}^{n} h_{i,j}(x_j^2 - x_j) \\ &= \sum_{i=1}^{t} \mathsf{mult}(f_i) + \sum_{j=1}^{n} \left(\sum_{i=1}^{t} h_{i,j} \right) (x_j^2 - x_j). \end{aligned}$$

Clearly $\sum_{i=1}^{t} \mathsf{mult}(f_i) = \mathsf{mult}(f)$. Further, if we define $h_j = \sum_{i=1}^{t} h_{i,j}$, then each h_j is computable by a sum of t ROABPs that have the required properties. ◀

We additionally require the following lemma from the work of Forbes, Shpilka, Tzameret, and Wigderson [21].

▶ **Lemma 4.5** ([21, Proposition 5.13])**.** *Let $\mathbb{F}$ be a field of characteristic zero, $X = \{x_0, \ldots, x_{2n-1}\}$, $T = \{t_{i,j} \ : \ i, j \in [0, \ldots, 2n-1] \text{ with } i < j\}$ be two sets of variables and $\beta > \binom{2n}{2}$ be any number. Further, let g be the unique multilinear polynomial such that*

$$g(X, T) \equiv \frac{1}{\sum_{i<j} t_{i,j} x_i x_j - \beta} \pmod{X^2 - X,\ T^2 - T}.$$

Then for any balanced partition (Y, Z) of X, $\mathrm{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] \geq 2^n$.

Finally, before we can prove our main theorem, we require a *weakness lemma* for a sum of multilinear ROABPs. The proof of this lemma is based on ideas which are implicitly present in the work of Chatterjee, Kush, Saraf, and Shpilka [17] (in the context of sums of ordered set-multilinear ABPs). However, we provide a detailed self-contained proof.

4.1 Weakness of a Sum of Multilinear ROABPs

Let $A = \sum_{i=1}^{t} A_i$ be a sum of ROABPs where each A_i is multilinear and computing some multilinear polynomial in $\mathbb{F}[x_1, \ldots, x_n]$. Without loss of generality, we can assume that each A_i has length n and say the maximum width is s. Since the polynomial is multilinear, the partial derivative matrix under any partition of variables (Y, Z) of X has dimension $2^{|Y|} \times 2^{|Z|}$ [5].

▶ **Fact 4.6.** *Let (Y, Z) be some partition of the variable set X and $M_{Y,Z}(f)$ be the partial derivative matrix with respect to this partition. Then,*

$$\mathrm{rank}[M_{Y,Z}(f)] \leq \min\{2^{|Y|}, 2^{|Z|}\} = 2^{\min\{|Y|,|Z|\}} \leq 2^{\frac{n-||Y|-|Z||}{2}}.$$

[5] The rows and columns are indexed by multilinear monomials over Y and Z variables respectively.

Next, we show that under any random balanced partition of the variables, the rank of any small size sum of multilinear ROABP reduces significantly with high probability.

▶ **Lemma 4.7** (Weakness Lemma). *Let $q = r = \sqrt{n}$. Further, let $A = \sum_{i=1}^{t} A_i$ be a sum of multilinear* ROABP*s computing a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ and (Y, Z) be a partition of the variable set X chosen independently and uniformly at random. Then, there exist constants $\varepsilon', \varepsilon'' \in (0, 1)$ such that*

$$\Pr_{(Y,Z)}\left[\text{rank}[M_{Y,Z}(A)] < t \cdot s^{q-1} \cdot 2^{\frac{n}{2} - \frac{\varepsilon' q\sqrt{r}}{4}} \mid (Y, Z) \text{ is balanced}\right] \geq 1 - t \cdot e^{-\varepsilon'' q}.$$

A proof of this can be found in the full version [15]. But we are now ready to prove an exponential lower bound against $\sum$ ROABP-IPS$_{\text{Lin}'}$ proofs.

4.2 Lower Bound over Fields of Characteristic Zero

▶ **Theorem 1.6.** *For $X = \{x_0, \ldots, x_{2n-1}\}$, $T = \{t_{i,j} : i, j \in [0, \ldots, 2n-1] \text{ with } i < j\}$ and $\beta = 2\binom{2n}{2}$, let $f \in \mathbb{F}[X, T]$ be the polynomial defined as*

$$f = \left(\sum_{0 \leq i < j \leq 2n-1} t_{i,j} x_i x_j\right) - \beta$$

which is unsatisfiable over the Boolean hypercube. Then there exists $\gamma > 0$ such that the total width of any sum of ROABP *computing the linear proof of unsatisfiability* (IPS$_{\text{Lin}'}$)*, for the system $\{f = 0, X^2 - X = 0, T^2 - T = 0\}$, is at least $\exp(n^\gamma)$.*

Proof. Let g be the unique multilinear polynomial that proves the unsatisfiability of f. Then, by Lemma 4.5, for any balanced partition (Y, Z) of X, $\text{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] \geq 2^n$. Thus,

$$\Pr_{(Y,Z)}\left[\text{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] < 2^n \mid (Y, Z) \text{ is balanced}\right] = 0.$$

Suppose g is computable by a sum of t multilinear ROABPs, say $A_1, \ldots A_t$, of width at most s. Since the total number of variables is $2n$, by Lemma 4.7, we have that for any partition (Y, Z) of X chosen uniformly at random,

$$\Pr_{(Y,Z)}\left[\text{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] < t \cdot s^{q-1} \cdot 2^{n - \frac{\varepsilon' q\sqrt{r}}{4}} \mid (Y, Z) \text{ is balanced}\right] \geq 1 - t \cdot e^{-\varepsilon'' q}.$$

for suitable constants $\varepsilon', \varepsilon'' \in (0, 1)$ and $q = r = \sqrt{2n}$.

Note that if $t > \exp(\varepsilon'' q)$, then we already have an $\exp(\Omega(\sqrt{n}))$ lower bound. Otherwise, there exists a balanced partition (Y, Z) such that $\text{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] \leq t \cdot s^{q-1} \cdot 2^{n - \frac{\varepsilon' q\sqrt{r}}{4}}$. However, we know that $\text{rank}_{\mathbb{F}(T)}[M_{Y,Z}(g)] \geq 2^n$. Hence, it must be the case that $t \cdot s^{q-1} \geq 2^{\varepsilon' q\sqrt{r}}$, which would imply that $s \geq \exp(n^{1/4})$. Either way, we get that the size of the sum of ROABPs computing g is at least $\exp(n^\gamma)$ for some $\gamma > 0$.

Let $h \in$ IPS$_{\text{Lin}}$, be any proof of unsatisfiability for f. Then $\mathsf{mult}(h) = g$. Assume, for sake of contradiction, that h can be computed by a sum of ROABPs, say $\sum A_i$, of size $\exp(o(n^\gamma))$.

$$h = \sum_i A_i \implies \mathsf{mult}(h) = \mathsf{mult}\left(\sum_i A_i\right) = \sum_i \mathsf{mult}(A_i)$$

Each ROABP $\mathsf{mult}(A_i)$ is a multilinear ROABP and using Proposition 4.4, $\sum_i \mathsf{mult}(A_i)$ is of size $\exp(o(n^{\gamma'}))$, but this contradicts the fact that any sum of multilinear ROABP computing $\mathsf{mult}(h) = g$ needs size at least $\exp(\Omega(n^\gamma))$. This completes the proof. ◀

4.3 Lower Bounds over Fields of Positive Characteristic

Note that in Theorem 1.6, the characteristic of $\mathbb{F}$ is required to be zero even though the weakness lemma (Lemma 4.7) for sum of multilinear-ROABPs is characteristic independent. This is because we use Lemma 4.5. So, in order to prove a theorem analogous to Theorem 1.6 in the positive characteristic setting, we need a statement analogous to Lemma 4.5 in this setting. The main idea for such a rank lower bound was recently shown in a work of Behera, Limaye, Ramanathan, and Srinivasan [8].

Using these ideas, we prove an exponential lower bound for the sum of ROABPs in positive characteristic. A proof sketch can be found in Subsection A.1.

▶ **Theorem 4.8.** *Let $n \in \mathbb{N}$ and p be any prime. Further let $\mathbb{F}'$ be a field of characteristic p with size p^{k+1}, where k is the smallest integer such that $p^k > \binom{2n}{2}2^{2n}$ and $\mathbb{F} \subset \mathbb{F}'$ be the subfield of size p^k. Also, arbitrarily fix $\beta \in \mathbb{F}' \backslash \mathbb{F}$.*

For $X = \{x_0, \ldots, x_{2n-1}\}$, $T = \{t_{i,j} \ : \ i, j \in \{0, \ldots, 2n-1 \text{ with } i < j\}\}$ and any $\bar{\alpha} \in \mathbb{F}^{\binom{2n}{2}}$, define

$$f = \left(\sum_{0 \leq i < j \leq 2n-1} \alpha_{i,j} t_{i,j} x_i x_j \right) - \beta$$

which is unsatisfiable over the Boolean hypercube. Then there exists $\bar{\alpha}$ and $\gamma > 0$, such that any sum of ROABP *computing the* $\mathsf{IPS}_{\mathsf{Lin}'}$ *refutation of the system $\{f = 0, X^2 - X = 0, T^2 - T = 0\}$ must have total width at-least $\exp(n^\gamma)$.*

In Theorem 4.8 the field size needs to be large since we are using Lemma A.1 to prove the rank lower bound. However, we can remove such a size requirement if we use the vector invariant polynomial from the work of Hakoniemi, Limaye, and Tzameret [24]. Details can be found in Subsection A.2.

Conclusion

The construction of the axiom polynomials from the subset-sum axiom polynomial is particularly relevant in proof complexity although the subset-sum polynomial is not directly a translation of CNFs. More details regarding this can be found in [22, Section 1.3]. If the axiom polynomial is sparse, then it can be directly constructed from the subset-sum polynomial by substituting monomials for the variables. In this paper, all the axiom polynomials except the one used in Theorem A.5 are sparse.

However, if one does not care about constructing sparse axiom polynomials, then the problem of proving polynomial-size quality lower bounds for formulas, ABPs, and circuits can be solved easily. This is noted in Observation B.2. As already mentioned above, it is more desirable if IPS lower bound results are shown for sparse axioms. This is also reflected if one compares the work of [3] with [22].

Finally, we state a few open problems for further study.

1. Prove (nearly) quadratic-size $\mathsf{IPS}_{\mathsf{Lin}'}$ lower bounds for formulas for an axiom polynomial which is sparse.
2. Prove super-linear size mult-$\mathsf{IPS}_{\mathsf{Lin}'}$ and $\mathsf{IPS}_{\mathsf{Lin}'}$ lower bounds for ABPs and circuits. Again, the axiom polynomials should be sparse.
3. Can we (re)-prove Theorem A.5 for a sparse axiom polynomial?

References

1 Miklós Ajtai. The complexity of the pigeonhole principle. In *29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 346–355. IEEE, 1988. doi:10.1109/SFCS.1988.21948.

2 Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, Las Vegas, Nevada, USA, October 14-17, 2001*, pages 190–199. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959893.

3 Robert Andrews and Michael A. Forbes. Ideals, determinants, and straightening: proving and using lower bounds for polynomial ideals. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 389–402. ACM, 2022. doi:10.1145/3519935.3520025.

4 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

5 Paul Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 23(4):740–751, 1994. doi:10.1137/S0097539791197240.

6 Paul Beame, Russell Impagliazzo, Jan Krajícek, Toniann Pitassi, and Pavel Pudlák. Lower bound on hilbert's nullstellensatz and propositional proofs. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 794–806. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365714.

7 Paul Beame, Toniann Pitassi, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. In *24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 200–220. ACM, 1992. doi:10.1145/129712.129739.

8 Amik Raj Behera, Nutan Limaye, Varun Ramanathan, and Srikanth Srinivasan. New bounds for the ideal proof system in positive characteristic. *Electron. Colloquium Comput. Complex.*, TR25-079, 2025. URL: https://eccc.weizmann.ac.il/report/2025/079/.

9 Michael Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86. ACM, 1983. doi:10.1145/800061.808735.

10 C. S. Bhargav, Prateek Dwivedi, and Nitin Saxena. Lower bounds for the sum of small-size algebraic branching programs. *Theor. Comput. Sci.*, 1041:115214, 2025. doi:10.1016/J.TCS.2025.115214.

11 Peter Bürgisser, Michael Clausen, and Mohammad A Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science & Business Media, 2013.

12 Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *J. Comput. Syst. Sci.*, 62(2):267–289, 2001. doi:10.1006/JCSS.2000.1726.

13 Samuel R. Buss, Russell Impagliazzo, Jan Krajícek, Pavel Pudlák, Alexander A. Razborov, and Jirí Sgall. Proof complexity in algebraic systems and bounded depth frege systems with modular counting. *Comput. Complex.*, 6(3):256–298, 1997. doi:10.1007/BF01294258.

14 Abhranil Chatterjee, Mrinal Kumar, and Ben Lee Volk. Determinants vs. algebraic branching programs. *Comput. Complex.*, 33(2):11, 2024. doi:10.1007/S00037-024-00258-Z.

15 Prerona Chatterjee, Utsab Ghosal, Partha Mukhopadhyay, and Amit Sinhababu. IPS lower bounds for formulas and sum of roabps. *CoRR*, abs/2507.09515, 2025. doi:10.48550/arXiv.2507.09515.

16 Prerona Chatterjee, Mrinal Kumar, Adrian She, and Ben Lee Volk. Quadratic lower bounds for algebraic branching programs and formulas. *Comput. Complex.*, 31(2):8, 2022. doi:10.1007/S00037-022-00223-8.

17 Prerona Chatterjee, Deepanshu Kush, Shubhangi Saraf, and Amir Shpilka. Lower bounds for set-multilinear branching programs. In Rahul Santhanam, editor, *39th Computational Complexity Conference, CCC 2024, July 22-25, 2024, Ann Arbor, MI, USA*, volume 300 of

LIPIcs, pages 20:1–20:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CCC.2024.20.

18 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

19 Tal Elbaz, Nashlen Govindasamy, Jiaqi Lu, and Iddo Tzameret. Lower bounds against the ideal proof system in finite fields. *arXive Preprint*, June 2025. Preprint. doi:10.48550/arXiv.2506.17210.

20 Michael A. Forbes. Low-depth algebraic circuit lower bounds over any field. In Rahul Santhanam, editor, *39th Computational Complexity Conference, CCC 2024, July 22-25, 2024, Ann Arbor, MI, USA*, volume 300 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CCC.2024.31.

21 Michael A. Forbes, Amir Shpilka, Iddo Tzameret, and Avi Wigderson. Proof complexity lower bounds from algebraic circuit complexity. *Theory Comput.*, 17:1–88, 2021. doi:10.4086/TOC.2021.V017A010.

22 Nashlen Govindasamy, Tuomas Hakoniemi, and Iddo Tzameret. Simple hard instances for low-depth algebraic proofs. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 188–199. IEEE, 2022. doi:10.1109/FOCS54457.2022.00025.

23 Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018. doi:10.1145/3230742.

24 Tuomas Hakoniemi, Nutan Limaye, and Iddo Tzameret. Functional lower bounds in algebraic proofs: Symmetry, lifting, and barriers. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1396–1404. ACM, 2024. doi:10.1145/3618260.3649616.

25 Johan Håstad. On small-depth frege proofs for PHP. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 37–49. IEEE, 2023. doi:10.1109/FOCS57990.2023.00010.

26 Johan Håstad and Kilian Risse. On bounded depth proofs for tseitin formulas on the grid; revisited. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1138–1149. IEEE, 2022. doi:10.1109/FOCS54457.2022.00110.

27 Pavel Hrubes and Amir Yehudayoff. Monotone separations for constant degree polynomials. *Inf. Process. Lett.*, 110(1):1–3, 2009. doi:10.1016/J.IPL.2009.09.003.

28 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the gröbner basis algorithm. *Comput. Complex.*, 8(2):127–144, 1999. doi:10.1007/S000370050024.

29 K. A. Kalorkoti. A lower bound for the formula size of rational functions. *SIAM Journal on Computing*, 14(3):678–687, 1985. doi:10.1137/0214050.

30 Jan Krajíček, Pavel Pudlák, and Alan Woods. Exponential lower bounds to the size of bounded depth frege proofs of the pigeonhole principle. *Random Structures & Algorithms*, 7(1):15–39, 1995. doi:10.1002/rsa.3240070103.

31 Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 2*, volume 2 of *Springer-Verlag Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, Heidelberg, 2005. doi:10.1007/b138062.

32 Fu Li, Iddo Tzameret, and Zhengyu Wang. Characterizing propositional proofs as noncommutative formulas. *SIAM J. Comput.*, 47(4):1424–1462, 2018. doi:10.1137/16M1107632.

33 Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 804–814. IEEE, 2021. doi:10.1109/FOCS52979.2021.00083.

34 Toniann Pitassi. Algebraic propositional proof systems. In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, January 14-17, 1996*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 215–244. DIMACS/AMS, 1996. doi:10.1090/DIMACS/031/07.

35 Toniann Pitassi and Russell Impagliazzo. Exponential lower bounds for the polynomial calculus. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 129–135. ACM, 1994. doi:10.1145/195058.195120.

36 Toniann Pitassi, Benjamin Rossman, Rocco A. Servedio, and Li-Yang Tan. Poly-logarithmic frege depth lower bounds via an expander switching lemma. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 644–657. ACM, 2016. doi:10.1145/2897518.2897637.

37 Alexander A. Razborov. Lower bounds for the polynomial calculus. *Comput. Complex.*, 7(4):291–324, 1998. doi:10.1007/S000370050013.

38 Robert A. Reckhow. *On the lengths of proofs in the propositional calculus*. Ph.d. thesis, University of Toronto, 1976. Technical Report #87.

39 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Manuscript online at https://github.com/dasarpmar/lowerbounds-survey/releases/download/v9.0.3/fancymain.pdf, 2021. A selection of lower bounds in arithmatic circuit complexity.

40 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

A Lower Bound Against Sum of ROABPs for $\mathsf{IPS}_{\mathsf{Lin}'}$ Proofs

A.1 Lower Bounds over Fields of Positive Characteristic of Large Size

We state the main lemma from the work of Behera, Limaye, Ramanathan, and Srinivasan [8].

▶ **Lemma A.1** ([8, Lemma 2.4]). *Let $\mathbb{F}, \mathbb{F}'$ be two fields such that $\mathbb{F} \subset \mathbb{F}'$, $n \in \mathbb{N}$ and X be the variable set. Fix $\beta \in \mathbb{F}' \setminus \mathbb{F}$ arbitrarily. Further, for any $\bar{\alpha} \in \mathbb{F}^n$ and a non-empty subset $S' \subseteq [n]$, let $g_{\bar{\alpha},S'}(\bar{x}) \in \mathbb{F}[X]$ be the unique multilinear polynomial that agrees with $\frac{1}{\sum_{i \in S'} \alpha_i \cdot x_i - \beta}$ over the Boolean hypercube.*

For any $S \subseteq \mathbb{F}$ which is finite, if we choose $\bar{\alpha}$ uniformly at random from S^n, then the following is true.

$$\Pr_{\bar{\alpha} \sim S^n} [\exists\ \emptyset \neq S' \subseteq [n] \ :\ \deg(g_{\bar{\alpha},S'}(\bar{x})) < |S'|] < \frac{2^{2n}}{|S|}$$

We will also need the following lemma from the work of Forbes, Shpilka, Tzameret, and Wigderson [21].

▶ **Lemma A.2** ([21, Lemma 5.12]). *Let $f \in \mathbb{F}[X, Y, Z]$ and $f_Z \in \mathbb{F}(Z)[X, Y]$ be the polynomial that symbolically equals f. That is, for any $\gamma \in \mathbb{F}^{|Z|}$, we have $f_\gamma(X, Y) = f(X, Y, \gamma) \in \mathbb{F}[X, Y]$.*

For any set of variables X, any field $\mathbb{F}$ and any $f \in \mathbb{F}[X]$, suppose $\mathsf{coeff}_X(f)$ is used to denote the coefficient vector of f. Then, for any $\gamma \in \mathbb{F}^{|Z|}$, $\dim_{\mathbb{F}(Z)}[\mathsf{coeff}_{X|Y}[f_Z(X, Y)]] \geq \dim_{\mathbb{F}}[\mathsf{coeff}_{X|Y}[f_\gamma(X, Y)]]$.

Using Lemma A.1 and Lemma A.2, Behera, Limaye, Ramanathan, and Srinivasan [8] prove a rank bound analogous to Lemma 4.5 in positive characteristic.

▶ **Lemma A.3** ([8, Lemma A.10]). *Let $n \in \mathbb{N}$ and $p \in \mathbb{N}$ be any prime. Say $\mathbb{F}'$ is a field of characteristic p with size p^{k+1}, where k is the smallest integer such that $p^k > \binom{2n}{n}2^{2n}$ and that $\mathbb{F}$ is a field of size p^k (so that $\mathbb{F} \subset \mathbb{F}'$). Fix $\beta \in \mathbb{F}'\setminus\mathbb{F}$ arbitrarily and finally, for any $\alpha \in \mathbb{F}^{\binom{2n}{2}}$, let $g_\alpha(\bar{x},\bar{t}) \in \mathbb{F}'[X,T]$ be the polynomial that agrees with $\frac{1}{\sum_{i<j}\alpha_{i,j}t_{i,j}x_ix_j-\beta}$ on the Boolean hypercube. Then there exists $\bar{\alpha}$ such that for any balanced partition (U,V) of X, $\operatorname{rank}_{\mathbb{F}'(T)}[M_{U,V}(g_\alpha)] \geq 2^n$.*

We are now ready to prove Theorem 4.8.

Proof. We combine Lemma A.3 with the weakness lemma (Lemma 4.7) to get the required lower bound. The argument works in a similar manner given in the proof of Theorem 1.6. ◀

A.2 Lower Bound over Fields of Characteristic At Least Five

We first state the rank lower bound from the work of Hakoniemi, Limaye, and Tzameret [24].

▶ **Lemma A.4** ([24, Lemma 43]). *Let $\mathbb{F}$ be a field of characteristic at least and $g(\bar{x},\bar{t}) \in \mathbb{F}[X,T]$ be the polynomial defined in Theorem A.5. Let $\hat{g}(\bar{x},\bar{t}) \in \mathbb{F}[X,T]$ be any polynomial that satisfies*

$$\hat{g}(\bar{x},\bar{t}) = \frac{1}{g(\bar{x},\bar{t})} \bmod (X^2 - X, T^2 - T).$$

If $\hat{g}_T(\bar{x}) \in \mathbb{F}(T)[X]$ is the polynomial that symbolically equals $\hat{g}$, then for any balanced partition (U,V) of X, $\operatorname{rank}_{\mathbb{F}(T)}[M_{U,V}(\hat{g}_T(X)] \geq 2^n$.

We are now ready to prove an exponential lower bound against $\sum$ ROABPs for proofs in $\mathsf{IPS}_{\mathsf{Lin}'}$ model when the field size is not necessarily large. The proof follows exactly along the lines of the one for Theorem 4.8 except that we use Lemma 4.7 combined with Lemma A.4 instead of Lemma A.3.

▶ **Theorem A.5.** *Let $\mathbb{F}$ be a field of characteristic at least 5. Further, let $X = \{x_1,\ldots,x_{4n}\}$ and $T = \{t_{i,j,k,l} \;:\; i,j,k,l \in [4n] \text{ with } i<j<k<l\}$ be two sets of variables.*

Define $g \in \mathbb{F}[X,T]$ to be the polynomial

$$g = \left(\prod_{1\leq i<j<k<l\leq 4n} 1 - t_{i,j,k,l} + t_{i,j,k,l}(x_ix_l - x_jx_k)\right) - \beta$$

which is unsatisfiable over Boolean hypercube as long as $\beta \neq \{-1,0,1\}$. Then there exists $\gamma > 0$ such that any sum of ROABP *computing a $\mathsf{IPS}_{\mathsf{Lin}'}$ refutation of the system $\{g = 0, X^2 - X = 0, T^2 - T = 0\}$ must have total width at-least $\exp(n^\gamma)$.*

B Some Related Observations

Now we sketch a few details that are noted in the conclusion. We start with the following fact, which is given in [21, 24].

▶ **Fact B.1.** *Let $n \geq d \in \mathbb{N}$ and $f = \mathrm{ESYM}_{n,d} - \beta$ for some $\beta \geq \binom{n}{d} \in \mathbb{F}$. Let $g(\bar{x})$ be the unique multilinear polynomial that agrees with $\frac{1}{f}$ over the Boolean hypercube. Then there exist $\beta' \neq 0$ and non-zero field elements $\alpha_d, \alpha_{d+1}, \ldots, \alpha_n$ such that $g(\bar{x}) = \sum_{i=d}^{n} \alpha_i \mathrm{ESYM}_{n,i} + \beta'$*

Since the polynomial $f(\bar{x})$ is symmetric and the multilinear polynomial $g(\bar{x})$ agrees with $\frac{1}{f(\bar{x})}$ over the Boolean hypercube, it must be the case that $g(\bar{x})$ is a symmetric multilinear polynomial. Hence, it can be expressed as a linear combination of elementary symmetric polynomials $\text{ESYM}_{n,i}$. The fact that the coefficients of $\text{ESYM}_{n,i}$ for $i < d$ are 0 follows from Lemma 3.2. Moreover, to the best of our knowledge, the best known formula upper bound for $g(\bar{x})$ is $O(dn^2)$. This follows by computing each $\text{ESYM}_{n,i}$ (for every $i \in [d, n]$) by a $O(n^2)$ size formula using Ben-Or's construction [9]. The circuit complexity of $g(\bar{x})$ is $O(n \log^2 n)$ [11] and ABP complexity of $g(\bar{x})$ is $O(n^2)$ (using Ben-Or's construction).

▶ **Observation B.2.** *Consider the polynomial $g(\bar{x})$ defined in Fact B.1 with $d = \frac{n}{10}$. Then the following statements are true.*

- *Any formula computing a* mult-IPS$_{\text{Lin}'}$ *refutation of the system $\{g = 0, X^2 - X = 0\}$ must have size at least $\Omega(n^2)$.*
- *Any ABP computing a* mult-IPS$_{\text{Lin}'}$ *refutation of the system $\{g = 0, X^2 - X = 0\}$ must have size at least $\Omega(n^2)$.*
- *Any circuit computing a* mult-IPS$_{\text{Lin}'}$ *refutation of the system $\{g = 0, X^2 - X = 0\}$ must have size least $\Omega(n \log n)$.*

Proof. By Proposition 2.13, if we show a $\Omega(n^2)$ formula (ABP) lower bound for the unique multilinear polynomial $f(\bar{X})$ that agrees with $\frac{1}{g(\bar{X})}$ over the Boolean hypercube, we get our mult-IPS$_{\text{Lin}'}$ refutation lower bound. By the Fact B.1, here $f(\bar{X}) = \text{ESYM}_{n,\frac{n}{10}}$ which has a $\Omega(n^2)$ formula (ABP) lower bound from [16]. Note that there is a crucial condition on the characteristic of field $\mathbb{F}$, $\text{Char}(\mathbb{F}) \nmid n$ in order to hold the lower bound. We do not know yet how to remove this condition. So, it is fair to assume $\text{Char}(\mathbb{F}) = 0$. For the circuit, we refer to the work [4], which shows any circuit computing $\text{ESYM}_{n,\frac{n}{10}}$ needs size at least $\Omega(n \log n)$. Here also the size of $\mathbb{F}$ needs to be either large enough or $\text{Char}(\mathbb{F}) = 0$ for the lower bound. ◀

Parallel Complexity of Depth-First-Search and Maximal Path in Restricted Graph Classes

Archit Chauhan[1]
Department of Computer Science and Engineering, IIT Bombay, India

Samir Datta
Chennai Mathematical Institute, India
UMI ReLaX, Indo-French joint research unit, Chennai, India

M. Praveen
Chennai Mathematical Institute, India
UMI ReLaX, Indo-French joint research unit, Chennai, India

Abstract

Constructing a Depth First Search (DFS) tree is a fundamental graph problem, whose parallel complexity is still not settled. Reif showed parallel intractability of lex-first DFS. In contrast, randomized parallel algorithms (and more recently, deterministic quasipolynomial parallel algorithms) are known for constructing a DFS tree in general (di)graphs. However a deterministic parallel algorithm for DFS in general graphs remains an elusive goal. Working towards this, a series of works gave deterministic NC algorithms for DFS in planar graphs and digraphs. We further extend these results to more general graph classes, by providing NC algorithms for (di)graphs of bounded genus, and for undirected H-minor-free graphs where H is a fixed graph with at most one crossing. For the case of (di)graphs of bounded treewidth, we further improve the complexity to a Logspace bound. Constructing a maximal path is a simpler problem (that reduces to DFS) for which no deterministic parallel bounds are known for general graphs. For planar graphs a bound of $O(\log n)$ parallel time on a CRCW PRAM (thus in NC^2) is known. We improve this bound to Logspace.

2012 ACM Subject Classification Theory of computation → Parallel algorithms

Keywords and phrases Parallel Complexity, Graph Algorithms, Depth First Search, Maximal Path, Planar Graphs, Minor-Free, Treewidth, Logspace

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.23

Related Version *Full Version*: `https://arxiv.org/pdf/2506.14974v2`

Funding *Samir Datta*: Partially supported by a grant from Infosys foundation.
M. Praveen: Partially funded by a grant from the Infosys foundation.

1 Introduction

Depth First Search (DFS) is an archetypal graph algorithm that has played a pivotal role in the study of several problems such as biconnectivity, triconnectivity, topological sorting, strong connectivity, planarity testing and embedding etc. Along with Breadth First Search (BFS) it is perhaps one of the best known algorithms. However, while the BFS algorithm is easy to parallelize, no such deterministic parallel algorithm is known for DFS[2]. In fact,

[1] Part of this work was done when the author was a PhD student at Chennai Mathematical Institute, India.

[2] When talking about parallel computation, we will mean the class NC. In terms of PRAM models, a problem is said to be in the NC if there exists a PRAM machine and constants k, c such that the machine solves the problem in $O(\log^k n)$ time, using $O(n^c)$ many parallel processors. If the machine is a Concurrent Read Concurrent Write PRAM, we denote this class by $\mathsf{CRCW}[\log^k n]$. An equivalent definition in terms of boolean circuits also exists. In particular $\mathsf{CRCW}[\log^k n]$ is equivalent to AC^k (see [39, Section 3.4]).


© Archit Chauhan, Samir Datta, and M. Praveen;
licensed under Creative Commons License CC-BY 4.0

Reif showed in the 1980's [48] that unless all of P is parallelizable, there is no parallel algorithm that can mimic the familiar recursive DFS. To be slightly more precise finding the lexicographically first DFS-order is P-complete. Roughly contemporaneously, the problem of finding a (not necessarily lexicographically first) DFS-order was reduced to the minimum weight perfect matching problem for bipartite graphs in [1] (and in [2] for directed graphs) which coupled with randomized parallel algorithm for min-weight perfect matching [40, 47], yielded a randomized parallel algorithm for DFS. Recent advances in matching algorithms [28] indeed yield a deterministic parallel algorithm for DFS although using quasipolynomially many (in particular, $n^{O(\log n)}$ many) processors.

A natural question to consider is: for which restricted graph classes can we give deterministic parallel algorithms for DFS. Planar graphs, as well as digraphs have been known to possess (deterministic) parallel DFS algorithms dating back to half a century ago [53, 33, 35, 38, 37, 3]. Khuller also gave an NC algorithm for DFS in $K_{3,3}$-minor-free graphs [41]. The above mentioned algorithms differ in their parallel complexity. For example Kao, Klein in [38] gave a parallel running time of $O(\log^{10} n)$ for DFS in planar digraphs while optimizing the number of required processors to be linear. More recently, Ghaffari et al [29] have given a (randomized) nearly work efficient algorithm for DFS running in $\tilde{O}(\sqrt{n})$ time and $\tilde{O}(m+n)$ work. Our metric for evaluating an algorithm is more complexity theoretic – so we try to optimise on the complexity class we can place the problem in. In particular, we are interested in placing the problems in the smallest possible class in the following (loose) hierarchy : $\mathsf{L} \subseteq \mathsf{UL} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^3 \subseteq \ldots$ (See Chapter 6 of [9]).

Most algorithms for parallel DFS in graphs proceed via finding a balanced path separator in the graph, which is a path such that removing all of its vertices leaves behind connected components (or strongly connected components in case of digraphs) of small size, allowing for logarithmic depth recursion. It was shown by Kao in [35] that all graphs and digraphs have balanced path separators, and it is easy to find one sequentially using the DFS algorithm itself. The challenge thus is to find a path separator in parallel. Another useful result Kao [36, 37] also showed is that if we have a small number of disjoint paths that together constitute a balanced separator (by small we mean up to polylogarithmically many, we call these *multi-path separators*), then we can trim and merge them into a single balanced path separator in parallel. If we could somehow merge and reduce a multi-path separator consisting of a large number of paths, say $O(n)$ many, to a multi-path separator consisting of up to constant (or even polylogarithmically many) number of paths in NC, then we would immediately have an NC algorithm for DFS, as the set of all vertices of the graph constitutes a trivial multi-path separator. Such a routine to reduce the number of paths in a multi-path separator is indeed the heart of the RNC algorithm of Aggarwal and Anderson [1] (and also [2]), and this is where the randomized routine for minimum weight perfect matching in bipartite graphs is used.

1.1 Our contributions

We first look at graphs of bounded genus in Section 4, which are a natural generalization of planar graphs (planar graphs have genus zero). We are not aware of any deterministic NC algorithm for DFS even for toroidal graphs. We give an NC algorithm for DFS which proceeds via finding a multi-path separator consisting of $O(g)$ many paths, where g is the genus of the graph. For graph classes of higher genus, the main bottleneck we encounter is that of testing their genus and finding an embedding into a surface of that genus in NC. More precisely, if we assume that we can find genus and embeddings of graphs of upto $O(\log n)$ genus in NC, then our algorithm still gives an NC bound for graphs of up to $O(\log n)$ genus.

In Section 5 we look at DFS in single crossing minor free graphs, which are another generalization of planar graphs (orthogonal to bounded genus graphs). By Wagner's theorem [58], we know that planar graphs are precisely those whose set of forbidden minors consists of K_5 and $K_{3,3}$. Both $K_5, K_{3,3}$ are single crossing graphs, i.e. they can be drawn on the plane with at most one crossing. We consider the family of H-minor-free graphs, where H is any fixed single crossing graph. Such families which include in particular the families of K_5-minor-free graphs and $K_{3,3}$-minor-free graphs, are called single-crossing-minor-free graphs. They form a natural stepping stone towards more general H-minor-free families, which by a deep theorem of Robertson and Seymour [51], completely characterize *minor-closed* families. The only result we know of in this direction is that of Khuller's [41] parallel algorithm for DFS in $K_{3,3}$-minor-free graphs. We generalize this result by giving an NC algorithm for DFS in single-crossing-minor-free graphs. The algorithm of [41] uses a decomposition of $K_{3,3}$-minor-free graphs by 2-*clique sums* (or separating pairs) into pieces that are either planar or are exactly the graph K_5. They proceed by finding the "center" piece of this decomposition tree and then finding a balanced cycle separator in it, after projecting the weights of attached subgraphs on the center piece. The high level strategy of our algorithm is similar. We first use a decomposition theorem [50] by Robertson and Seymour which states that single-crossing-minor-free graphs can be decomposed by 3-*clique sums* into pieces that are either planar or of bounded treewidth. Then we try to find a balanced cycle separator in the center piece. However the attachment at 3-cliques instead of 2-cliques poses some technical difficulties, as simply projecting the weights of attached subgraphs on the center piece does not work. To deal with this, we use appropriate gadgets while projecting, that mimic the connectivity of the attached subgraphs. We remark here that the problem of finding a minimum weight perfect matching was shown to be in (deterministic) NC for bounded genus bipartite graphs [19, 32] as well as for single-crossing-minor-free graphs [27]. However, the reductions in [1, 2] involve adding large bi-cliques in the input graph, which does not seem to preserve the genus, nor the forbidden minors of the graph.

In Section 6 we look at bounded treewidth (di)graphs, for which we are not aware of any explicitly stated parallel algorithms for DFS. We note however that the machinery of Kao [35] easily yields an $\mathsf{AC}^1(\mathsf{L})$ (which is contained in AC^2. See Section 2.) algorithm for DFS in bounded treewidth graphs. We improve this bound to L. Instead of using path separators, our technique for bounded treewidth (di)graphs involves the parallel (in fact, Logspace) version of Courcelle's theorem from [25]. We also use an existing result [17, 11] that states that we can add a linear ordering to the vertices of the graph without blowing up the treewidth of the structure. This combined with an observation of [23] characterizing the lex-first DFS tree in terms of lex-first paths to each vertex, allows us to express membership of an edge to the DFS tree, in MSO.

Finally, in Section 7 we turn to the problem of finding a maximal path in a planar graph, or MaximalPath. Given an input (undirected) graph and a vertex r in it, the problem is to find a (simple) path starting at r that cannot be extended further, i.e., all the neighbours of its end point other than r already lie on the path. This problem easily reduces to DFS because every root to leaf path in a DFS-tree is a maximal path. However even for this problem, no deterministic parallel algorithm for general graphs is known, and the RNC algorithm by [7] goes via reduction to perfect matching. For restricted graph classes, the situation is clearer for MaximalPath as compared to DFS. Anderson and Mayr [8] gave a deterministic $O(\log^3 n)$ time algorithm on a CREW PRAM, for graphs of bounded degeneracy [3] These include for

[3] A graph is said to be k-degenerate if every subgraph of it has a vertex of degree at most k.

example, planar graphs, and even more generally, all H-minor free graph classes. In modern terms, their algorithm witnesses a bound of $\mathsf{AC}^1(\mathsf{L})$. A better bound of $\mathsf{CRCW}[\log n]$ for MaximalPath in *planar graphs* comes from the DFS algorithm of Hagerup [33], which again can be improved by a modern analysis to $\mathsf{UL} \cap \mathsf{co\text{-}UL}$ [4]. We improve this bound to L. Our algorithm proceeds by first reducing the problem to finding a maximal path in the triconnected components of a planar graph using the 2-clique sum decomposition (also referred to as the SPQR decomposition) of [21]. We then show that in the triconnected planar components, a maximal path can be found in the "outerplanar layers" of the graph. The results we prove can be summarized as follows:

▶ **Theorem 1.** *We have the following bounds:*

1. DFS *in bounded genus graphs and digraphs is in* $\mathsf{AC}^1(\mathsf{UL} \cap \mathsf{co\text{-}UL})$.
2. DFS *in undirected single-crossing-minor-free graphs is in* NC.
3. DFS *in directed and undirected graphs of bounded treewidth is in* L
4. MaximalPath *in undirected planar graphs is in* L.

2 Preliminaries

We assume the reader is familiar with tree-decompositions, planarity, embeddings on surfaces and graph minors. When referring to concepts like treewidth, minors and genus of digraphs, we will always intend them to apply to the underlying undirected graph. In digraphs, we will use the term path to mean simple directed path unless otherwise stated. We will use *tail* of a path to refer to the starting vertex of the path and *head* of a path to refer to the ending vertex of the path. The term *disjoint paths* will always mean vertex disjoint paths, and the term *internally disjoint paths* will refer to paths that are disjoint except possibly sharing end points. In a given graph $G = (V, E)$ and edge $(u, v) \in E(G)$, we denote the graph obtained by removing the edge (u, v) (but keeping the vertices u, v intact) by $G - (u, v)$.

We refer the reader to [9] for definitions of classes $\mathsf{L}, \mathsf{NL}, \mathsf{AC}, \mathsf{NC}$. A nondeterministic Turing machine is said to be unambiguous if for every input, there is at most one accepting computation path. The class UL is the class of problems for which there exists an unambiguous Logspace Turing machine to decide the problem. In our setting, our outputs are not binary, but encodings of objects like a DFS tree, or a maximal path. Each of the classes mentioned above can be generalised to deal with *functions* instead of just *languages* in a natural way. One may refer to [34] or [3, Section 2] for more details, specially what is meant by the computation of functions by a bounded space class like L, NL or $\mathsf{UL} \cap \mathsf{co\text{-}UL}$. In simplest terms, the output has a size of polynomially many bits and each bit can be computed in the stated class. We will often refer to a machine that computes a function as a transducer. A constant number of L transducers may be composed in L (see [9, Chapter 4]). A similar result holds for classes like $\mathsf{NL}, \mathsf{UL} \cap \mathsf{co\text{-}UL}$ (see [3, Section 2] for a more elaborate explanation).

An important result in the Logspace world is that of Reingold [49] which states that the problem of deciding reachability (as well as its search version) in undirected graphs is in L. Using this and the lemma of composition of L transducers, many routines like computing connected components left after removing some vertices, or counting the number of vertices in a connected component, can be shown to be in L. Another result we will use is that the problem of reachability in directed planar graphs is in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$, as shown by [13, Theorem 1.1]. This extends to graphs of bounded genus, and in fact to graphs of $O(\log n)$ genus if their polygonal schema is given as part of input [31, Theorem 7] (See also [43, 54, 19]). The theorem of [31] achieves this by giving a logspace algorithm to compute $O(\log n)$ size weight functions for edges that isolate shortest paths between vertices of the input graph. In

other words, for any pair of vertices u, v, there is a unique path minimum weight path from u to v under the weight function computed by [31, Theorem 7]. Thierauf and Wagner [55] showed that with such path isolating weights assigned to edges, we can compute distances as well as shortest path between vertices, in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$. These results will be useful in computing arborescences[4] in bounded genus graphs in Section 4.

Suppose $\mathcal{C}$ is a complexity class (a set of boolean functions). We use $\mathsf{AC}^k(\mathcal{C})$ to denote functions that can be computed by a family of AC^k circuits augmented with oracle gates that can compute functions in $\mathcal{C}$. Note for example, that since $\mathsf{UL} \cap \mathsf{co\text{-}UL} \subseteq \mathsf{AC}^1$, the class $\mathsf{AC}^1(\mathsf{UL} \cap \mathsf{co\text{-}UL})$ is contained in AC^2.

We will use the notion of Gaifman graph of a logical structure:

▶ **Definition 2.** *Given a logical structure $\mathcal{A}$, with a finite universe $|\mathcal{A}|$, and relation symbols which are interpreted as relations of $\mathcal{A}$, the Gaifman graph of $\mathcal{A}$, $G_{\mathcal{A}}$ is a graph whose vertices are the elements of $|\mathcal{A}|$, and for $u, v \in |\mathcal{A}|$, we add an edge between them iff u, v both occur in a relation tuple of $\mathcal{A}$.*

The treewidth of the structure is the treewidth of its Gaifman graph. We will use the following extension of Courcelle's theorem by [25]:

▶ **Theorem 3** ([25]). *For every $k \geq 1$ and every MSO-formula ϕ, there is a logspace DTM that on input of any logical structure $\mathcal{A}$ of treewidth at most k, decides whether $\mathcal{A} \models \phi$ holds.*

Clique-Sum decompositions

We recall the notion of *clique-sums*. Given two graphs G_1, G_2, a k-clique-sum of them can be obtained from the disjoint union of G_1, G_2 by identifying a clique in G_1 of at most k vertices with a clique of the same number of vertices in G_2, and then possibly deleting some of the edges of the merged clique. It is used in reverse while decomposing a graph by clique sums. For our purpose, we will use up to 3-clique sum decomposition of a graph. Suppose G decomposes via a 3-clique sum at clique c into G_1 and G_2. Then we write G as $G_1 \oplus_c G_2$. More generally, if $G_1, G_2, \ldots, G_\ell$ all share a common clique c, then we use $G_1 \oplus_c G_2 \oplus_c \ldots \oplus_c G_\ell$ to mean $G_1, G_2, \ldots, G_\ell$ are glued together at the shared clique. When separating the graph along a separating pair/triplet, we add virtual edges if needed to make the separating pair/triplet a clique. The decomposition can be thought of as a two colored tree (see [14, 27, 16]) where the blue colored nodes represent *pieces* (subgraphs that the graph is decomposed into), and the red colored nodes represent the cliques at which two or more pieces may be attached. The edges of the clique describe the incidence relation between pieces and cliques (see Figure 2). For single-crossing-minor-free graphs, Robertson and Seymour gave the following theorem regarding their 3-clique sum decomposition:

▶ **Theorem 4** ([50]). *For any single-crossing graph H, there is an integer τ_H such that every graph with no minor isomorphic to H can be obtained by 3-clique sums, starting from planar graphs and graphs of treewidth at most τ_H.*

An observation we use is that we can assume that in any planar piece of the decomposition, the vertices of a separating pair or triplet lie on a common face (Otherwise we could decompose the graph further. See [14, 27] for a more detailed explanation). It is shown in [27] that the decomposition can be computed in NC. For computing the 2-clique sum decomposition, we

[4] An arborescence of a digraph G is a spanning tree rooted at a vertex r, such that all edges of the tree are directed away from r.

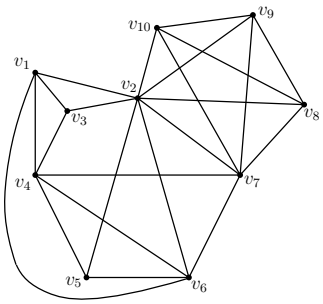


Figure 1 An example of a graph G. We ignore directions here.

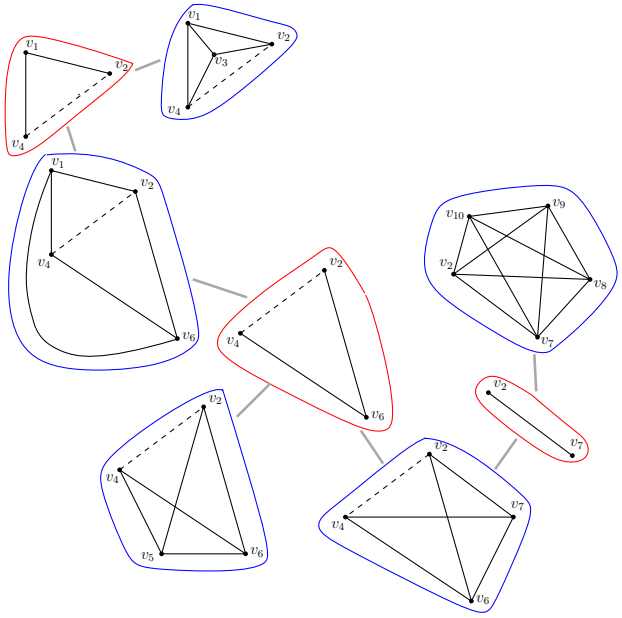


Figure 2 A clique sum decomposition of G. Red nodes are the clique nodes and blue node the piece nodes. Dashed edges denote virtual edges.

will use the Logspace algorithm given by [21]. The resulting pieces of the decomposition, also called as the *triconnected components* of the graph, are either dipoles, simple cycles, or non-trivial 3-connected graphs (i.e. 3-connected graphs with at least four vertices).

Balanced Path/Cycle separators

Consider an input graph $G = (V, E)$ with $|V| = n$. If G is undirected, we use the term α-separator (or just balanced separator) in general to mean some subgraph of G which if removed from G, then the size of each of the resulting connected components is at most αn, where $0 < \alpha < 1$. A single path P, or a collection of paths $\{P_1, \ldots P_k\}$ for example could form a balanced separator of G. We call the latter a *multi-path* separator consisting of k paths. If G is directed, we have a similar definition, but the constraint is instead on the size of each *strongly* connected component left after removing the vertices of the separator. Note that if we have an algorithm to find an α-separator in a directed graph, then we can use that to find an α-separator in an undirected graph by making each edge into a bi-directed edge. Any simple cycle C in an embedded planar graph G will divide it into two regions, the *interior* and *exterior* of C, which we denote by $int(C)$ and $ext(C)$ respectively. Typically, if a face of G is marked as the outer face then the region containing the outer face is referred to as the exterior and the other region as the interior. Suppose the vertices, edges and faces of G are assigned weights. Then we use $w(int(C))$ to denote the sum of weights of vertices, edges, faces that are contained in $int(C)$. The term $w(ext(C))$ is defined similarly. The vertices, edges that lie on C itself are usually not counted since if they are part of the separator and will be removed when recursing. Suppose u, v are two vertices in the cycle C (embedded in the plane). We denote by $u \circlearrowright v$ (respectively $u \circlearrowleft v$), the segment of C obtained by starting from u and going clockwise (respectively counter-clockwise) along C till v.

3 Path/Cycle Separators review

We restate some of the existing algorithms and analyze their complexity with respect to circuits/space bounded complexity classes. First, we reiterate an algorithm of Kao (see also [35, 2, 37]) that combines a small number of path separators into a single path separator in parallel, and observe that it is in L given an oracle for REACH. We state the theorems and corollaries for digraphs, but the corresponding statements for undirected graphs also hold with appropriate changes.

▶ **Theorem 5** ([36, 37]). *Let G be a digraph and $\{P_1, P_2\}$ be two paths that together form an α-separator of G for some $\alpha \in [\frac{1}{2}, 1)$. Given an oracle for* REACH*, we can in* L*, construct a single path P_3 that is also an α-separator of G.*

We reproduce the proof for completeness in Section A.1.1. Note that as a corollary, the above algorithm is in NL for general directed graphs, and in UL ∩ co-UL for directed graphs of $O(\log n)$ genus (using [31]). For undirected graphs, the algorithm (with appropriate changes) is in L using Reingold's algorithm [49]. As noted in [2, 38], if we have a multi-path separator consisting of k many paths, say $\{P_1, P_2 \dots P_k\}$, we can combine them into a single path separator using k iterations of the above procedure with k transducers. Thus we have:

▶ **Corollary 6.** *Let G be a digraph and k be a constant. Let $\{P_1, P_2, \dots P_k\}$ be k disjoint paths that form a multi-path α-separator of G for some $\alpha \in [\frac{1}{2}, 1)$. Given an oracle for* REACH*, we can in* L*, construct a single path P_{k+1} that is also an α-separator of G.*

For bounded treewidth graphs, we can find a tree decomposition in L using [25, Theorem 1.1], and we know that there is always a bag in the tree decomposition of a graph whose vertices together form a 1/2-separator of the graph. Since REACH is known to be in L for digraphs of bounded treewidth (from Theorem 3), we have the following result:

▶ **Corollary 7.** *Given a digraph of bounded treewidth, we can find a 1/2-path separator in* L*.*

It is shown in [3, Section 7] how to construct a DFS tree in a digraph given a routine for finding path separators, in parallel. The algorithm follows a divide and conquer strategy, where we find DFS trees on the smaller strongly connected components in parallel, and sew them together using a routine for DFS in DAGs. We state the result in the following proposition.

▶ **Proposition 8.** *Let G be a digraph and $r \in V(G)$. Let $F_{sep}(H)$ be a function that takes as input any digraph H and returns a balanced path separator of H. Then there exists a uniform family of AC^1 circuits with oracle gates for F_{sep} and functions computable in* UL ∩ co-UL*, that take as input (G, r), and return a DFS tree of G rooted at r.*

Although the proposition is for digraphs, the same idea works in the simpler setting of undirected graphs also (Instead of DAGs of strongly connected components, we just have to recurse on connected components there. See for example [53]). We will also use the following, more general notion of cycle separators in planar graphs, that was introduced in [45]:

▶ **Definition 9.** *Let G be an undirected planar graph, and $w\colon \{V(G) \cup E(G) \cup F(G)\} \to \mathbb{Q}^+$ be a normalised weight function assigning weights to vertices, edges and faces of G that sum up to 1. We call a cycle C a* 2/3- interior-exterior *cycle separator if $w(int(C)) \leq 2/3$, and $w(ext(C)) \leq 2/3$.*

Miller showed in [45] that such a weighted cycle separator always exists if G is biconnected and if no face is too heavy. Moreover, we can find such a separator in parallel[5] (see also [52]).

▶ **Theorem 10** ([45]). *Let G be a biconnected planar graph, and $w\colon \{V(G)\cup E(G)\cup F(G)\} \to \mathbb{Q}^+$ be an assignment of non-negative weights to vertices, edges, faces that sum up to 1, such that no face has weight more than 2/3. Then we can construct a* 2/3-interior-exterior-*cycle separator of G in* CRCW$[\log n]$.

4 DFS in directed graphs of bounded genus

We show how to find balanced path separators (and therefore a DFS tree) in digraphs that can be embedded on a surface of genus at most a fixed constant g, in NC. For an undirected graph, it is easy to see that a DFS tree of the bidirected version of it is also a DFS tree of the original graph.

The first step naturally is to find an embedding, which can be done using the theorem of Elberfeld and Kawarabayashi [26, Theorem 1], which states that we can find an embedding of a graph of genus at most a constant g, on a surface of genus g, in Logspace. The idea is to find a small number of disjoint, surface non-separating cycles (not necessarily directed cycles), say $C_1, C_2, \ldots C_l$ ($l \leq g$) in G, such that the (weakly) connected components left after removing them are all planar. These can be found with at most g sequential iterations. Then, if any of the remaining planar components has a large strongly connected component, we can find a 2/3-path separator, say P, in it using [38, Theorem 19]. A more modern analysis of the algorithm in [38] gives a UL ∩ co-UL bound (see [15, Section 3.3] for a proof. See also [3, Lemma 50], which gives an 11/12-path separator with the same bound). Then $C_1, C_2 \ldots C_l, P$ together form a balanced multi-path separator of the digraph G. The (possibly undirected) cycles $C_1, C_2 \ldots C_l$ will have an additional property. Any $C_i \in \{C_1, C_2 \ldots C_l\}$ will either be a directed cycle, or it will consist of two internally disjoint (directed) paths that share a common tail vertex and a common head vertex. Therefore the multi-path separator can be written as consisting of at most $2g + 1$ many paths, and we can use Corollary 6 to get the required path separator. We use the following known fact (see [46, Lemma 4.2.4] or [30, Fact 9.1.6]).

▶ **Theorem 11** ([46]). *Let G be a graph embeddable on a surface $\mathcal{S}$, of genus g. Let C be a cycle that forms a surface non-separating curve on some embedding of G on $\mathcal{S}$. Then every connected component of $G - C$ is embeddable on a surface of genus $g - 1$.*

Therefore, if we have an algorithm to find such a (undirected) cycle in NC, then we can repeat it g times to find the required (undirected) cycles $C_1, C_2 \ldots C_k$. We will use the following lemma from [5] (See also [56]) to find such cycles.

▶ **Lemma 12** ([5]). *Let G be a graph of genus $g > 0$, and let T be a spanning tree of G. Then there is an edge $e \in E(G)$ such that $T \cup e$ contains a surface non-separating cycle.*

Clearly the theorem also works if we have an arborescence instead of a spanning tree. The fundamental cycle C corresponding to a non-tree edge $e = (u, v)$ would consist of the non-tree edge e and two vertex disjoint directed paths, say P, P', starting from a common ancestor of (u, v) in the arborescence (we can include the common ancestor vertex in either of P or P'), and ending at u, v respectively. We can find an arborescence in UL ∩ co-UL using [31, 55] (as

[5] The theorem in [45] also gives a bound on size of the separator but that is not important for us.

explained in the appendix), and the required fundamental cycle in Logspace using the theorem of [26]. We describe the steps of the algorithm in detail in Section A.2. Therefore we can find a 2/3-path separator in directed graphs of bounded genus in UL ∩ co-UL. Using Proposition 8, we can find a depth first search tree in such graphs in AC^1(UL ∩ co-UL), finishing the proof of part 1 of Theorem 1.

5 DFS in Single-Crossing-Minor-Free Graphs

We start with computing the 3-clique-sum decomposition by the NC algorithm of [27]. We will denote this decomposition tree by T_G, each node of which is either a piece node or a clique node, and the weight of each node is the number of vertices in the corresponding piece/clique. (Note that by this convention, the sum of weights of all nodes of T_G will sum up to more than n since some vertices would occur in multiple pieces).

▶ **Definition 13.** *Let t_R be a node of T_G, and R its corresponding piece/clique, such that the size of any connected component in $G - R$ is less than $n/2$. We call t_R a central node of T_G, and R a central piece/clique.*

Such a node always exists and can be found in Logspace by traversing along the heaviest child in T_G (see Chapter 7 of [18] for example). We will assume t_R to be the root of T_G hereafter. We will denote the *subtrees* of T_G that are children of t_R by $\{T_1, T_2, \dots T_l\}$, the subgraphs of G corresponding to these subtrees by $\{G_1, G_2, \dots G_l\}$, and the cliques by which they are attached to R by $\{c_1, c_2 \dots c_l\}$ respectively. These subgraphs might themselves consist of smaller subgraphs glued at the common clique. For example, G_1 might consist of $G_{11}, G_{12}, \dots G_{1k}$ that are glued at the shared clique c_1 (i.e. $G_1 = G_{11} \oplus_{c_1} G_{12} \dots \oplus_{c_1} G_{1k}$). See Figure 3 for reference. Note that because R is a central node, $w(G_{1i}) - w(c_1) < n/2$ $\forall i \in [1..k]$. If t_R is a clique node, then we have a 1/2-separator of at most three vertices and we are done. Therefore there are two cases to consider. Either R is a planar piece, or a piece of bounded treewidth.

When R is of bounded treewidth

In this case, we will refine the tree T_G by further decomposing R. Since R is of treewidth at most τ_H, we can compute a tree decomposition of R such that every bag is of size at most $\tau_H + 1$, in L using [25]. Let this tree be denoted by T_R. Consider the subtree T_1 of T_G as described above, which has a node t_{c_1} by which it is attached to R. Since c_1 is a clique, there must be at least one bag in T_R that contains all vertices of c_1 (see [12]). Attach t_{c_1} to any such node of T_R. Do this procedure for all subtrees $T_1, T_2, \dots T_l$. It is easy to see that this will result in another tree decomposition, and that at least one of the nodes of T_R will be a central node of this new tree. Hence we can use the procedure described above to find it in L. Since the bags of T_R are of size at most $\tau_H + 1$, we get a $1/2-$*separator* of G consisting of constant number of vertices, and we are done.

When R is planar

We start with a simple idea, but encounter an obstacle. Then we show how to fix it. Note that R (along with its virtual edges), is biconnected. To initialize, we assign weight 1 to all vertices of R and weight 0 to all edges and faces of R. We project the weight of subgraphs $G_1, G_2 \dots G_l$ on vertices, edges and faces of R, at the respective cliques they are attached at. Suppose the subgraph G_k is attached to R at clique c_k. Then we project the weight as follows.

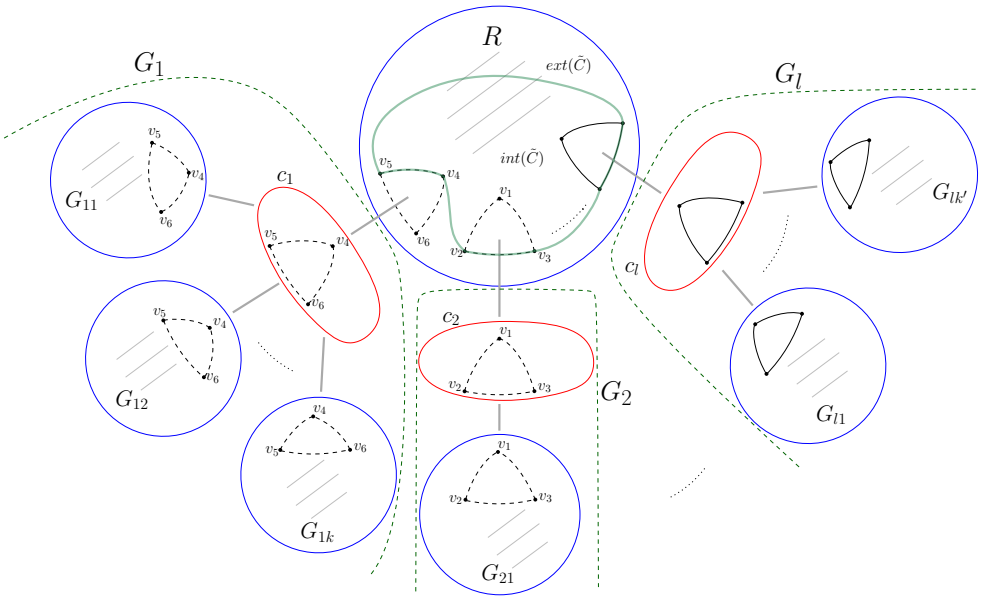


Figure 3 R is the central node with children subgraphs $G_1, G_2, \ldots G_l$ attached to it via cliques $c_1, c_2, \ldots c_l$ respectively. G_1 consists of subgraphs $G_{11}, G_{12} \ldots G_{1k}$ glued at c_1. G_2 is same as G_{21}. Dashed edges denote virtual edges in R. In the case when R is a planar piece, it is useful to think of it as a sphere, with separating triplets forming empty triangular faces in R, on which $G_1, G_2, \ldots G_l$ are attached. The cycle in green is an example of *interior-exterior*-cycle separator $\widetilde{C}$ (though gadget replacements are not shown in the figure). Of the attached subgraphs shown, G_2, G_l are *interior components* with respect to $\tilde{C}$ whereas G_1 is an *exterior component* with respect to $\tilde{C}$.

1. If c_k is a single vertex v, then assign a weight equal to $|V(G_k)|$ to *vertex* v.
2. If c_k is a separating pair (u, v), then we assign a weight equal to $|V(G_k)| - 2$ to the *edge* (u, v) of R. (We subtract 2 since weights of vertices u, v are already accounted for).
3. If c_k is 3-clique of vertices (v_1, v_2, v_3), then we assign weight equal to $|V(G_k)| - 3$ to the *face* of R enclosed by c_k.

Let $\widetilde{R}$ denote the weighted version of R obtained after projecting these weights. The sum of weights of all vertices, edges, faces of R is n. There are two possibilities. Suppose there exists a clique, say c_1 (which is either a vertex, or an edge, or a face of $\widetilde{R}$), with weight at least $2n/3$. This means that total weight of subgraphs attached to R via c_1, i.e $w(G_{11} \oplus_{c_1} G_{12} \oplus_{c_1} \ldots \oplus G_{1k})$, is at least $2n/3$. Since each of the graphs $G_{11}, G_{12} \ldots G_{1k}$ is of size at most $n/2$ and they all get disconnected on removal of c_1, the vertices of c_1 form a $1/2-$*separator* of G, and we are done. Therefore the remaining possibility we need to consider is one where every vertex, edge, face of $\widetilde{R}$ has weight smaller than $2n/3$. Since $\widetilde{R}$ is planar and biconnected, we can find a $2/3$-*interior-exterior*-cycle-separator of it in L using Theorem 10. Let $\widetilde{C}$ be such a separator. Label one of the regions $\widetilde{C}$ divides R into as the interior of $\widetilde{C}$ and the other as the exterior of $\widetilde{C}$. Observe that for any subgraph that is attached to R (like G_2 for example), its vertices of attachment to R (the clique via which it is attached), will either all lie on or in the interior of $\widetilde{C}$, or all lie on or in the exterior of $\widetilde{C}$. We call the attached subgraphs of the former kind *interior components* of R with respect to $\widetilde{C}$, and the latter as *exterior components* of R with respect to $\widetilde{C}$ (see Figure 3). We will drop the phrase "with respect to $\widetilde{C}$" for brevity. On removing the vertices of $\widetilde{C}$ in G, there cannot remain any path in G from any vertex of an interior component of R to any vertex of an exterior component of R. Therefore we have the following claim:

▷ Claim 14. Let $\widetilde{C}$ be an α-*interior-exterior*-cycle-separator of $\widetilde{R}$, where $\alpha \in [\frac{1}{2}, 1)$. Then the vertices of $\widetilde{C}$ form an α-separator of G.

Now if we could replace the virtual edges of $\widetilde{C}$ by valid path segments in the attached subgraphs then we would get a simple cycle C in G which would be a $2/3-$*separator* of G, and we would be done. However there is an issue here, as the virtual edges in a 3-clique need not capture *"disjointness"* of the external connections accurately. For example, suppose $c_2 = \{v_1, v_2, v_3\}$ is a 3-clique and the subgraph G_2 is attached to R at c_2 (as shown in Figure 3). Suppose (v_1, v_2), and (v_1, v_3) are virtual edges in R, and our cycle separator $\widetilde{C}$ uses both of these edges (they must necessarily be consecutive in $\widetilde{C}$). Though there must be paths between v_1, v_2, and between v_1, v_3 in G_2 (since it is attached via 3-clique), there might not exist such paths that are internally disjoint. Hence instead of just projecting the weight of G_2 on the face formed by c_2 in R, we use a more appropriate gadget to mimic the connectivity of G_2.

Gadgets for the subgraphs attached to R via 3-cliques

The gadget for an attached graph like G_2 is essentially a coarser planar projection of its block-cut tree, that captures the needed information regarding the cut vertices associated with its terminals $\{v_1, v_2, v_3\}$. We can adversarially assume that all three edges $(v_1, v_2), (v_2, v_3), (v_3, v_1)$ are virtual edges.

▶ **Definition 15.** *Let G_2 be a graph and vertices $v_1, v_2, v_3 \in V(G_2)$ be called its terminals. The* disjoint path configuration *of G_2 with respect to its terminals v_1, v_2, v_3 consists of three boolean variables : $DP(v_1\text{-}v_2\text{-}v_3), DP(v_2\text{-}v_3\text{-}v_1), DP(v_3\text{-}v_1\text{-}v_2)$. $DP(v_1\text{-}v_2\text{-}v_3)$ takes value* True *if there exists a path between v_1 and v_3 that goes via v_2, and* False *otherwise. $DP(v_2\text{-}v_3\text{-}v_1), DP(v_3\text{-}v_1\text{-}v_2)$ are defined similarly.*

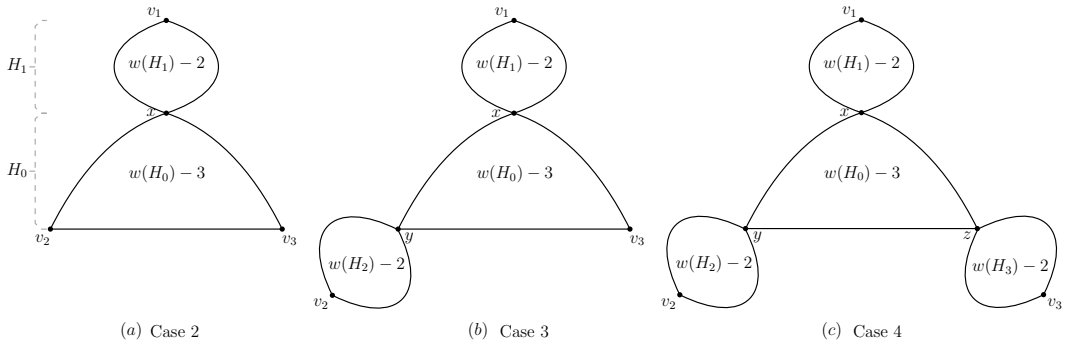
Their are four cases we consider for possible disjoint path configurations, others are handled by symmetry.

	$DP(v_1\text{-}v_2\text{-}v_3)$	$DP(v_2\text{-}v_3\text{-}v_1)$	$DP(v_3\text{-}v_1\text{-}v_2)$
Case 1 :	*True*	*True*	*True*

This means G_2 is biconnected, and hence we do not need a new gadget. We just keep the virtual edges between the clique vertices and assign weight equal to $|V(G_2)| - 3$ to the face enclosed by them (as described in Item 3). If $\widetilde{C}$ takes two virtual edges of c_2, we can use the two disjoint path algorithm in [42] to find the corresponding paths in G_2.

	$DP(v_1\text{-}v_2\text{-}v_3)$	$DP(v_2\text{-}v_3\text{-}v_1)$	$DP(v_3\text{-}v_1\text{-}v_2)$
Case 2 :	*True*	*True*	*False*

By Menger's theorem (see [44], or Chapter 3 of [24]), we know that there must be at least one cut vertex in G_2 that separates $\{v_1\}$ from $\{v_2, v_3\}$. Let x be a cut vertex separating these. Let the connected component of $G_2 - x$ containing v_1, augmented with x, be H_1. Let the component containing v_2, v_3, augmented with x, be H_0 (i.e. $V(H_1) \cap V(H_0) = \{x\}$ and $V(H_1) \cup V(H_0) = V(G_2)$). We choose x such that there is no cut vertex in H_0, that separates $\{v_1\}$ and $\{v_2, v_3\}$ in G_2 (i.e. x is the cut vertex described that is "furthest" from v_1). It is easy to see that x is unique. By our choice of x, there must be a path in H_0 from v_2 to v_3 via x. Thus the gadget G_2' is as shown in Figure 4, containing vertices v_1, v_2, v_3, x, and the total weight distributed among the faces as shown. The following lemma shows the correctness of this procedure.



(a) Case 2 (b) Case 3 (c) Case 4

Figure 4 Figure (a) shows the gadget for graph G_2 for Case 2, (b) for Case 3 and (c) for Case 4. The faces formed by cycles of the gadget corresponding to the subgraphs H_0, H_1, H_2, H_3 of G_2 are assigned the weight functions as shown. The total weight in each gadgets sums up to $w(G_2)$. The subgraph H_0 of G_2 has no cut vertex in each case. In the gadget for Case 4, it is possible that the block H_0 is empty, i.e. there exists a vertex ($x = y = z$) in G_2 which on removal disconnects all of v_1, v_2, v_3 from each other. We don't draw that gadget separately since its structure is clear.

▶ **Lemma 16.** *Suppose G_2, which is the subgraph attached to R via c_2 has the disjoint path configuration as described in Case 2. Suppose it is replaced in $\widetilde{R}$ by the gadget G_2' defined above for this case. Then if an α-*interior-exterior*-cycle separator $\widetilde{C}$ of $\widetilde{R}$ uses any edges of this gadget, we can replace them by paths in G_2 such that the new cycle C is a simple cycle and it remains an α-separator in $R \oplus G_2$.*

Proof. We defer the proof to the full version of this paper. ◀

Case 3 :	$DP(v_1\text{-}v_2\text{-}v_3)$	$DP(v_2\text{-}v_3\text{-}v_1)$	$DP(v_3\text{-}v_1\text{-}v_2)$
	False	*True*	*False*

The gadget is shown in Figure 4. As in the previous case, we first find the "furthest" vertex x separating $\{v_2, v_3\}$ and $\{v_1\}$. Then, in the component containing v_2, v_3 (augmented with x), we find the cut vertex y separating $\{x, v_3\}$ and $\{v_2\}$, that is furthest from v_2.

Case 4 :	$DP(v_1\text{-}v_2\text{-}v_3)$	$DP(v_2\text{-}v_3\text{-}v_1)$	$DP(v_3\text{-}v_1\text{-}v_2)$
	False	*False*	*False*

Like in previous cases, we find the "furthest" cut vertices x, y, z that separate v_1, v_2, v_3 from rest of graph. There are two cases, either there exists a single cut vertex w such that $G_2 - w$ disconnects all of v_1, v_2, v_3 from each other, which is essentially the case when $x = y = z = w$. Other case is when there is no such cut vertex. The gadget is shown in Figure 4. The proof of correctness of gadgets for Cases 3,4 is similar to the proof of Lemma 16 presented in the full version.

To find these cut vertices, we can in parallel check if each vertex of G_2 satisfies the properties of vertices "x, y, z" described above. Thus the gadgets can be constructed in NC. We described the procedure for G_2, but the construction can be done in parallel for all subgraphs attached to R via 3-cliques. After finding a 2/3-*interior-exterior*-cycle separator of $\widetilde{R}$ separator using Theorem 10, we can, in parallel, replace the virtual/gadget edges by acutal paths in the attached subgraphs to get the cycle separator of G. In the blocks of gadgets where we need to find a path between two vertices via a third vertex, we can use the 2-disjoint path algorithm described in [42]. Thus we can construct a 2/3-path-separator of G in NC. Combined with Proposition 8, this finished the proof of part 2 of Theorem 1.

6 DFS in bounded treewidth directed graphs in L

The input graph G can be seen as a structure $\mathcal{A}$ whose universe, $|\mathcal{A}|$ is $V(G)$, and its vocabulary consists of binary relation symbol E which is interpreted as the edge relation $E(G)$ over $V(G)$. The Gaifman graph of $\mathcal{A}$, $G_{\mathcal{A}}$, is isomorphic to G. We will need to add a linear order O on $|\mathcal{A}|$, which would amount to adding $|V(G)|$ many edges in $G_{\mathcal{A}}$. A known result in model theory (see [11, Lemma 1], also [17, Theorem VI.4]) is that we can add a linear order to a structure, such that the treewidth of the structure is increased by at most a constant. It is shown in [10, Lemma 5.2] that we can construct such a linear order in L. Therefore after computing the order O and adding it to $\mathcal{A}$ the new structure $\mathcal{A}'$ also has bounded treewidth. Moreover, for any two vertices u, v, we can query if u is "lesser than" v in the transitive closure of O, in logspace. We also add the edges $E(G)$ to our universe of discourse, as well as an incidence relations : $tail(e, v)$ which is true iff the edge $e \in E(G)$ has vertex v as its tail, and $head(e, v)$ defined similarly. This can again be done without blowing up the treewidth (see section 6 of the full version).

We use $u <_O v$ to denote that vertex u is lesser than vertex v in the transitive closure of O. An ordering over vertices defines a unique lexicographically first DFS tree. The algorithm will go over each edge of E, and use Theorem 3 to query if it belongs to the lex-first DFS tree of G, in L. Our universe is $V \cup E$, and the relations on it given are $tail, head$, and the linear order O. To express the membership of an edge in the lex-first DFS tree with respect to O, we will use a theorem connecting the lex-first DFS tree to lex-first paths. We define lexicographic ordering on paths starting from a common vertex:

▶ **Definition 17.** *Let P_1, P_2 be two paths in G starting from r. We say that P_1 is lexicographically lesser than P_2 (with respect to O) if at the first point of divergence starting from r, P_1 diverges to a vertex that is lesser in the given ordering than the one P_2 diverges to. We denote this as $P_1 <_O P_2$.*

This naturally defines the notion of the lexicographically least path starting from r and ending at a vertex v. We call it the lex-first path from r to v. We use the following result which is shown in the proof of Theorem 11 in [23]:

▶ **Theorem 18.** *Let T_l be the lex-first DFS tree of G with respect to a given linear order O. Then $\forall v \in V(G)$, the unique path from r to v in T_l is exactly the lex-first path from r to v (with respect to O) in G.*

Thus in order to check if an edge (u, v) belongs to T_l (rooted at r), it suffices to check if it is the last edge of the lex-min path in G from r to v. We define a formula $\phi_{DFS}(u, v)$, which is true iff the edge (u, v) is part of the lex-first DFS tree T_l. Thus the logspace algorithm is the following : For each edge (u, v) in E, query $\phi_{DFS}(u, v)$ and add to output tape iff the query returns yes. A DFS numbering or order of traversal of the vertices of T_l can easily be obtained from T_l by doing an Euler tree traversal of T_l with respect to the ordering O. Now all that remains is to express the formula $\phi_{DFS}(u, v)$ in MSO. We defer that to the full version.

7 Maximal Paths in planar graphs in L

In this section we will show that given a planar undirected graph G and a vertex r in it, we can find a maximal path in G starting from r, in L. We will assume that our graph is embedded in a plane with a face marked as outer face (a planar embedding can be found in

L by [6, 22]). The first step is to reduce the problem to that of finding a maximal path in a triconnected component of G that is 3-connected (i.e. it is not a cycle or a dipole). We decompose the graph into its triconnected components, which can be done in L as described in [20]. We can choose a leaf piece L of this decomposition tree which is connected to its parent piece H by separating pair, say (r_0, r_1). The basic idea is to find a path from r (L is chosen such that it does not contain r) to one of the vertices of the separating pair, say r_0, without going through r_1. Then it suffices to find a maximal path in L starting from r_0, that does not end at r_1. Some care must be taken in the reduction however, and cases like when L is a simple cycle need to be handled. We defer the details of the reduction to the full version.

7.1 Maximal Paths in 3-connected components

By the reduction above, we can assume that our input graph is a 3-connected component of G, say L, and our goal is to find a maximal path in L starting at r_0 and not ending at r_1. Note that r_0, r_1 lie on a common face of L, and the edge (r_0, r_1) might be a virtual edge in L. Hence we will in general look at the graph $L' = L - (r_0, r_1)$, that is, the graph obtained from L by removing the virtual edge (r_0, r_1) (not the vertices). Let the outer face of L, on which both r_0, r_1 lie, be called f_0. The outer face of L' is naturally obtained by merging f_0 with the other face adjacent to (r_0, r_1). Let this be called f_0'. We first note the following observation:

▶ **Observation 19.** *The boundary of the face f_0' of L' is a simple cycle.*

If not, there would be a cut vertex x in L'. Then (r_0, x) would form a separating pair of L, contradicting the fact that it is 3-connected. We hereby denote this cycle, that is the boundary of L', by C. We recall the definition of bridges of a cycle in a graph, which (roughly) are components of the graph minus the cycle, along with their attachments to the cycle.

▶ **Definition 20** ([57])**.** *For a subgraph H of G, a vertex of attachment of H is a vertex of H that is incident with some edge of G not belonging to H. Let J be an undirected cycle of G. We define a bridge of J in G as a subgraph B of G with the following properties:*
1. *each vertex of attachment of B is a vertex of J .*
2. *B is not a subgraph of J.*
3. *no proper subgraph of B has both the above properties.*

A bridge could also be a chord of C. We call such bridges as *trivial* bridges, and others as *non-trivial* bridges. We need the following observation for L' and C:

▶ **Observation 21.** *Any non-trivial bridge B of C (in the graph L'), if one exists, must have at least three vertices of attachment.*

This holds because if a non-trivial bridge of C in L' had only two vertices of attachment on C, they would form a separating pair of L.

There is a natural ordering we can give to the vertices of attachments of a bridge, that is the order in which they are encountered while travelling C, say clockwise, starting from r_0. We now define the *span* of a bridge.

▶ **Definition 22.** *Let the vertices of cycle C be $\langle u_0 = r_0, u_1, u_2 \dots u_l \rangle$ in clockwise order in the given embedding. Let B be a bridge of C with vertices of attachment $u_{i_0}, u_{i_1}, \dots u_{i_k}$, where $0 \leq i_0 < i_1 \dots i_k \leq l$. Then the set of vertices $\{u_{i_0}, u_{i_0+1}, \dots u_{i_k}\}$ is called the* span *of bridge*

B with respect to r_0 (or just span *of B for brevity), denoted by $span(B)$. The set of vertices $\{u_{i_k}, u_{i_k+1} \dots u_l, u_0, \dots u_{i_0}\}$ is called the* complement span *of bridge B (with respect to r_0). We say that bridge B has minimal span if for any other bridge B' of C, $span(B') \nsubseteq span(B)$.*

Basically, the vertices of attachment of a bridge divide C into segments. The span of a bridge consists of vertices of C, exluding the ones in the segment containing r_0, while the complement span consists of the segment containing r_0. To disambiguate the case when r_0 itself is a vertex of attachment of a bridge B, note that the definition uses the convention of putting the segment adjacent to r_0 on the clockwise side into the span of B, and the segment adjacent to r_0 on the counter clockwise side into the complement span of B. Since G is planar and C is outer boundary, the spans of bridges form a laminar family.

▶ **Observation 23.** *Let B_1, B_2 be two bridges of C, lying in the interior of C. Then either $span(B_1) \subset span(B_2)$, or $span(B_2) \subset span(B_1)$, or $span(B_1) \cap span(B_2) = \emptyset$.*

Now we define the *second-leftmost* path of a bridge of C.

▶ **Definition 24.** *Let B be a non-trivial bridge of cycle C, with vertices of attachment $u_{i_0}, u_{i_1}, u_{i_2} \dots u_{i_k}$ according to the ordering specified above. The* second leftmost *path of B starting from u_{i_0}, is the one obtained by the following procedure:*

- *Initialize u_{i_0} as current vertex and u_{i_0+1} as its* parent*.*
- *At any step, let the current vertex be v, and its parent be w. Let z be the next vertex occuring after w* other than the vertex u_{i_1}*, in the clockwise ordering of neighbours of v in the embedding. Travel to z and make it the current vertex. Make v its parent.*
- *Repeat the above step, until either a vertex repeats in the traversal, or a vertex of C (after the initial occurence of u_{i_0}) is reached.*

We denote this path by $P_{sl}(B, u_{i_0})$.

We next state a lemma that the second-leftmost path of a non-trivial bridge, starting at u_{i_0}, is indeed a simple path that ends at u_{i_2} because L is 3-connected.

▶ **Lemma 25.** *Given a non-trivial bridge B of cycle C, with vertices of attachment $u_{i_0}, u_{i_1}, u_{i_2} \dots u_{i_k}$ as described above, the following hold:*

1. *$P_{sl}(B, u_{i_0})$ is a simple path that ends at u_{i_2}.*
2. *All neighbours of u_{i_1} that lie in B, also lie on $P_{sl}(B, u_{i_0})$.*

Proof. We defer the proof to the full version. ◀

Since we know that $P_{sl}(B, u_{i_0})$ is a simple path starting and ending at C, it is clear that we can perform the traversal to construct it in L, by just storing the current vertex, the parent vertex, and the vertex u_{i_1} at each step. The reason we choose the *second leftmost path* in the bridge is so that after branching from the cycle C into its interior at u_{i_0}, we can emerge back at C at u_{i_2}, and then trap the maximal path in the segment $u_{i_2} \circlearrowleft u_{i_0}$ of C. The existence of the unvisited vertex u_{i_1} ensures that the segment is non-empty and we can do so. In the case when the bridge B is a chord, we define its second leftmost path as just the chord itself.

We now describe the algorithm to compute maximal path in L starting from r_0, with a small exception that we will handle subsequently.

1. Compute the bridges of C (w.r.t. L') by computing the components of $G - C$ along with their vertices of attachment to C.
2. From $r_0 = u_0$, extend the path by going clockwise till we find the first point of attachment of a bridge B of C with minimal span. Let the vertices of attachment of B, in clockwise order starting from r_0 be $u_{i_0}, u_{i_1}, u_{i_2}, \dots u_{i_k}$ (where $i_k \geq 2$). If B is a chord, we denote its vertices of attachment by u_{i_0}, u_{i_2}.

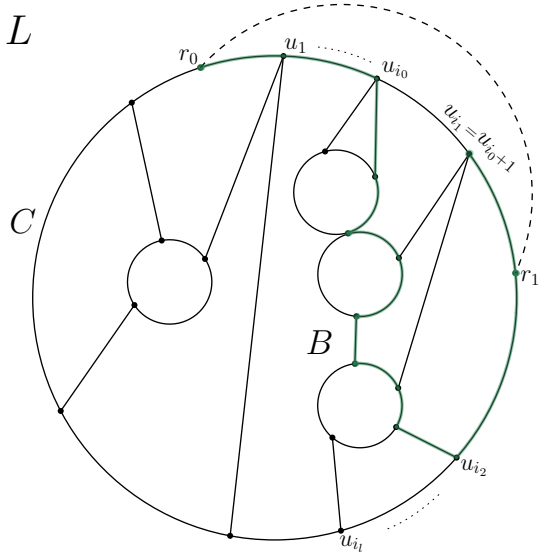


Figure 5 The 3-connected graph L. The dashed edge between r_0, r_1 is a virtual edge. The boundary of $L - (r_0, r_1)$ is the cycle C. The bridge on the right side is the labelled as B, with vertices of attachment $u_{i_0}, u_{i_1}, u_{i_2}, \ldots u_{i_l}$. The maximal path is highlighted in green, starting from r_0, going till u_{i_0}, then taking the second leftmost path in B to reach u_{i_2}, and then encircling back towards u_{i_0}, stopping at its neighbour u_{i_0+1} (In this figure, u_{i_1} is same as u_{i_0+1}).

3. From u_{i_0}, continue along the path $P_{sl}(B, u_{i_0})$ till we reach u_{i_2} (we will reach u_{i_2} by Lemma 25). From u_{i_2}, extend this path by traversing the segment $u_{i_2} \circlearrowleft u_{i_0+1}$, till the vertex u_{i_0+1}. Suppose that $u_{i_0+1} \neq r_1$. Then this path is a maximal path.

To see that this path is maximal (assuming $u_{i_0+1} \neq r_1$), note that no other bridge of C can be enclosed inside $P_{sl}(B, u_{i_0})$ and the segment $u_{i_2} \circlearrowleft u_{i_0}$ of C, since we chose B with a minimal span. There are two possible cases, either u_{i_0+1} has no neighbour in the interior of C in which case we are done. The only other possiblity, by the above argument is that B is a non-trivial bridge and $u_{i_0+1} = u_{i_1}$ (B cannot be a chord between u_{i_0} and u_{i_0+1} as we assume input graph to be simple). In this case, by Lemma 25, its neighbours inside C are all visited in $P_{sl}(B, u_{i_0})$, and we are done.

To handle the case when $u_{i_0+1} = r_1$, we can apply the same algorithm, but some operations "reversed" (like taking the bridge with minimal *complement span*, and taking the second *rightmost* path etc). We defer this argument to the full version.Each of the steps in the algorithm like computing bridges, checking for minimal span, can be done in L using constantly many transducers and [49]. And as explained above, computing the required segments of C, and the second leftmost path, can also be done in L.

References

1 Alok Aggarwal and Richard J. Anderson. A random NC algorithm for depth first search. *Comb.*, 8(1):1–12, 1988. doi:10.1007/BF02122548.

2 Alok Aggarwal, Richard J. Anderson, and Ming-Yang Kao. Parallel depth-first search in general directed graphs. *SIAM J. Comput.*, 19(2):397–409, 1990. doi:10.1137/0219025.

3 Eric Allender, Archit Chauhan, and Samir Datta. Depth-first search in directed planar graphs, revisited. *Acta Informatica*, 59(4):289–319, 2022. Preliminary version appeared in MFCS 2021. doi:10.1007/S00236-022-00425-1.

4 Eric Allender, Archit Chauhan, Samir Datta, and Anish Mukherjee. Planarity, exclusivity, and unambiguity. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:39, 2019.

5 Eric Allender, Samir Datta, and Sambuddha Roy. The directed planar reachability problem. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2005. doi:10.1007/11590156_19.

6 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117–134, 2004. doi:10.1016/j.ic.2003.09.002.

7 Richard Anderson. A parallel algorithm for the maximal path problem. *Combinatorica*, 7(4):315–326, 1987. doi:10.1007/BF02579320.

8 Richard J. Anderson and Ernst W. Mayr. Parallelism and the maximal path problem. *Inf. Process. Lett.*, 24(2):121–126, 1987. doi:10.1016/0020-0190(87)90105-0.

9 Sanjeev Arora and Boaz Barak. *Computational Complexity, a modern approach*. Cambridge University Press, 2009.

10 Nikhil Balaji. Succinct numbers, skew circuits and bounded treewidth graphs: Algorithms and complexity. In *PhD Thesis, Chennai Mathematical Institute*, 2016. URL: https://scholar.google.co.in/scholar?oi=bibs&cluster=8516246372270295301&btnI=1&hl=en.

11 Nikhil Balaji and Samir Datta. Bounded treewidth and space-efficient linear algebra. In Rahul Jain, Sanjay Jain, and Frank Stephan, editors, *Theory and Applications of Models of Computation*, pages 297–308, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-17142-5_26.

12 Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science*, pages 1–10, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

13 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, 2009. doi:10.1145/1490270.1490274.

14 Erin Wolf Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *Journal of Graph Algorithms and Applications*, 17(3):201–220, 2013. doi:10.7155/jgaa.00291.

15 Archit Chauhan. Some problems on planar and minor free graphs. In *PhD Thesis, Chennai Mathematical Institute*, 2024. URL: https://libarchive.cmi.ac.in/theses/archit_chauhan_cs2024.pdf.

16 Archit Chauhan, Samir Datta, Chetan Gupta, and Vimal Raj Sharma. The even-path problem in directed single-crossing-minor-free graphs. In *To appear in 49th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, LIPIcs, 2024. doi:10.4230/LIPIcs.MFCS.2024.43.

17 Yijia Chen and Jörg Flum. On the ordered conjecture. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 225–234. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.33.

18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

19 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N.V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012. In Commemoration of Amir Pnueli. doi:10.1016/j.jcss.2011.11.002.

20 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 203–214, 2009. doi:10.1109/CCC.2009.16.

21 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. *ACM Trans. Comput. Theory*, 14(2):8:1–8:33, 2022. doi:10.1145/3543686.

22 Samir Datta and Gautam Prakriya. Planarity testing revisited. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation - 8th Annual Conference, TAMC 2011, Tokyo, Japan, May 23-25, 2011. Proceedings*, volume 6648 of *Lecture Notes in Computer Science*, pages 540–551. Springer, 2011. doi:10.1007/978-3-642-20877-5_52.

23 Pilar de la Torre and Clyde P. Kruskal. Fast parallel algorithms for all-sources lexicographic search and path-algebra problems. *J. Algorithms*, 19(1):1–24, 1995. doi:10.1006/jagm.1995.1025.

24 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2016.

25 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

26 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *ACM Symposium on Theory of Computing (STOC)*, pages 383–392, 2014. doi:10.1145/2591796.2591865.

27 David Eppstein and Vijay V. Vazirani. Nc algorithms for computing a perfect matching and a maximum flow in one-crossing-minor-free graphs. *SIAM Journal on Computing*, 50(3):1014–1033, 2021. doi:10.1137/19M1256221.

28 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. A deterministic parallel algorithm for bipartite perfect matching. *Commun. ACM*, 62(3):109–115, 2019. doi:10.1145/3306208.

29 Mohsen Ghaffari, Christoph Grunau, and Jiahao Qu. Nearly work-efficient parallel dfs in undirected graphs. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 273–283, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558481.3591094.

30 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017. doi:10.1017/9781139028868.

31 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Reachability in o(log n) genus graphs is in unambiguous logspace. 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 34:1–34:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.STACS.2019.34.

32 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Efficient isolation of perfect matching in o(log n) genus bipartite graphs. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 43:1–43:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.MFCS.2020.43.

33 Torben Hagerup. Planar depth-first search in O(log n) parallel time. *SIAM J. Comput.*, 19(4):678–704, June 1990. doi:10.1137/0219047.

34 B. Jenner and B. Kirsig. Alternierung und Logarithmischer Platz. Dissertation, Universität Hamburg, 1989.

35 Ming-Yang Kao. All graphs have cycle separators and planar directed depth-first search is in DNC. In John H. Reif, editor, *VLSI Algorithms and Architectures, 3rd Aegean Workshop on Computing, AWOC 88, Corfu, Greece, June 28 - July 1, 1988, Proceedings*, volume 319 of *Lecture Notes in Computer Science*, pages 53–63. Springer, 1988. doi:10.1007/BFb0040373.

36 Ming-Yang Kao. Linear-processor NC algorithms for planar directed graphs I: strongly connected components. *SIAM J. Comput.*, 22(3):431–459, 1993. doi:10.1137/0222032.

37 Ming-Yang Kao. Planar strong connectivity helps in parallel depth-first search. *SIAM J. Comput.*, 24(1):46–62, 1995. doi:10.1137/S0097539792227077.

38 Ming-Yang Kao and Philip N. Klein. Towards overcoming the transitive-closure bottleneck: Efficient parallel algorithms for planar digraphs. *Journal of Computer and System Sciences*, 47(3):459–500, 1993. doi:10.1016/0022-0000(93)90042-U.

39 Richard M. Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD-88-408, University of California, Berkeley, March 1988. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html.

40 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Comb.*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

41 Samir Khuller. Extending planar graph algorithms to k_3,3-free graphs. *Inf. Comput.*, 84(1):13–25, 1990. doi:10.1016/0890-5401(90)90031-C.

42 Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. Processor efficient parallel algorithms for the two disjoint paths problem and for finding a kuratowski homeomorph. *SIAM J. Comput.*, 21(3):486–506, 1992. doi:10.1137/0221032.

43 Jan Kyncl and Tomás Vyskocil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Trans. Comput. Theory*, 1(3):8:1–8:11, 2010. doi:10.1145/1714450.1714451.

44 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. URL: http://eudml.org/doc/211191.

45 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

46 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001. URL: http://jhupbooks.press.jhu.edu/ecom/MasterServlet/GetItemDetailsHandler?iN=9780801866890&qty=1&source=2&viewMode=3&loggedIN=false&JavaScript=y.

47 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

48 John H. Reif. Depth-first search is inherently sequential. *Inf. Process. Lett.*, 20(5):229–234, 1985. doi:10.1016/0020-0190(85)90024-9.

49 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291.

50 N. Robertson and P.D. Seymour. Excluding a graph with one crossing. *Graph struc- ture theory (Seattle, WA, 1991)*, 1993. doi:10.1090/conm/147.

51 Neil Robertson and P.D. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte. doi:10.1016/j.jctb.2004.08.001.

52 Gregory E. Shannon. A linear-processor algorithm for depth-first search in planar graphs. *Information Processing Letters*, 29(3):119–123, 1988. doi:10.1016/0020-0190(88)90048-8.

53 Justin R. Smith. Parallel algorithms for depth-first searches i. planar graphs. *SIAM J. Comput.*, 15(3):814–830, 1986. doi:10.1137/0215058.

54 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012. doi:10.1016/j.ic.2012.03.002.

55 Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theory Comput. Syst.*, 47(3):655–673, 2010. doi:10.1007/s00224-009-9188-4.

56 Carsten Thomassen. Embeddings of graphs with no short noncontractible cycles. *Journal of Combinatorial Theory, Series B*, 48(2):155–177, 1990. doi:10.1016/0095-8956(90)90115-G.

57 W. T. Tutte. Separation of vertices by a circuit. *Discrete Mathematics*, 12(2):173–184, 1975. doi:10.1016/0012-365X(75)90032-1.

58 Klaus Von Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937. URL: https://api.semanticscholar.org/CorpusID:123534907.

A Appendix

A.1 Appendix for Section 3

A.1.1 Proof of Theorem 5

We restate the theorem here:

▶ **Theorem 5** ([36, 37]). *Let G be a digraph and $\{P_1, P_2\}$ be two paths that together form an α-separator of G for some $\alpha \in [\frac{1}{2}, 1)$. Given an oracle for* REACH*, we can in* L*, construct a single path P_3 that is also an α-separator of G.*

Proof. Let the two paths that together form an α separator be $P_1 = \{u_1, u_2 \dots u_p\}$ and $P_2 = \{v_1, v_2 \dots v_q\}$.

- Trim the path P_1 to $P_1' = \{u_1, u_2 \dots u_s\}$ $(s \leq p)$ such that separator property is just lost on trimming away u_{s-1}. That is, $P_1' \cup P_2$ is an α-separator of G, but $P_1' \backslash \{u_s\} \cup P_2$ is not. This means that in $G - (P_1' \backslash \{u_s\} \cup P_2)$, the vertex u_s is a part of a strongly connected component of size at least αn.
- Having trimmed P_1 to P_1', repeat the same step and trim P_2 to $P_2' = \{v_t, v_{t+1} \dots v_q\}$ $(t \geq 1)$ such that $P_1' \cup P_2'$ forms an α-separator, and the vertex v_t is part of some strongly connected component of size at least αn in the graph $G - (P_1' \cup P_2' \backslash \{v_t\})$.
- Thus in $G - (P_1' \backslash \{u_s\} \cup P_2' \backslash \{v_t\})$, there are two strongly connected components of size at least αn $(\alpha \geq 1/2)$, containing u_s, v_t resepectively. This means that they must be part of a single strongly connected component and hence there must be a path P from u_s to v_t in $G - (P_1' \backslash \{u_s\} \cup P_2' \backslash \{v_t\})$. Therefore we can concatenate the paths P_1', P, P_2' into a single path $P_1'.P.P_2'$, which is an α-separator of G.

While trimming paths in steps 1 and 2, we only need to remember the current end point of the trimmed path, which can be done in L. Other steps, like checking the size of leftover connected components and finding a path from u_s to v_t in the subgraph can be done in L given an oracle to REACH. ◀

A.2 Appendix for Section 4

We describe the steps of the algorithm for computing path separators in bounded genus digraphs here:

1. Decompose the graph into strongly connected components. If there is no strongly connected component of size larger than $2n/3$, then we are done, else let G_0 be the component larger than $2n/3$.
2. Find an embedding of G_0 using [26].
3. Find an arborescence of G_0. To do this, first construct the path isolating weigths for edges using [31, Theorem 7] so that there is a unique min weight path between any two vertices of G_0. Then for every vertex, we find the shortest path from a vertex say r_0 to it in UL $\cap$ co-UL using [55] (see Section 2). This will clearly give an arborescence of G_0 rooted at r_0.
4. By Lemma 12, we known that there exists an edge such that its fundamental cycle, is a surface non-separating cycle. To find it we can go over every edge, compute its fundamental cycle C and use the theorem of [26] to check if every weakly connected component of $G_0 - C_1$ has genus strictly lesser than that of G_0. Let C_1 denote the surface non-separating cycle we found. This can be done in Logspace using transducers.

5. Let G_1 be the largest strongly connected component of $G_0 - C_1$. If it is more than $2n/3$ in size then we do the steps above on G_1. Repeat until finally we have graph $G_{rem} = G - (C_1 \cup C_2 \dots C_l)$ (where $l < g$) such that either all strongly connected components of G_{rem} are smaller than $2n/3$ or all the weakly connected components are planar. In the latter case, find a 2/3-path separator P in the strongly connected component in G_{rem} that is of size more than $2n/3$.
6. For each cycle C_i we found at each step, let P_i, P_i' denote the two directed paths that form the cycle C_i as explained in Section 4. The paths $P_1 \cup P_1' \dots P_l \cup P_l' \cup P$ (P if required) together form a 2/3-separator of G.
7. Use Corollary 6 to merge the paths $P_1, P_1', \dots P_l, P_l', P$ into a single path P_s which is also a 2/3-path separator of G. Output P_s.

Steps 1 to 4 can be done in L or in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$. Step 5 involves at most g iterations of steps 1 to 4 and is therefore in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$ as the class is closed under composition for constant number of transducers. The merging of at most $2l + 1$ paths into a single path separator is in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$ by Corollary 6. Therefore the entire algorithm to construct the separator is in $\mathsf{UL} \cap \mathsf{co\text{-}UL}$ as it uses a constant number of transducers.

Languages of Words of Low Automatic Complexity Are Hard to Compute

Joey Chen
Department of Mathematics, National University of Singapore, Singapore

Bjørn Kjos-Hanssen
Department of Mathematics, University of Hawai'i at Mānoa, Honolulu, HI, USA

Ivan Koswara
School of Computing, National University of Singapore, Singapore

Linus Richter[1]
Department of Mathematics, National University of Singapore, Singapore

Frank Stephan
Department of Mathematics, National University of Singapore, Singapore
School of Computing, National University of Singapore, Singapore

Abstract

The automatic complexity of a finite word (string) is an analogue for finite automata of Sipser's distinguishing complexity (1983) and was introduced by Shallit and Wang (2001). For a finite alphabet Σ of at least two elements, we consider the non-deterministic automatic complexity given by *exactly* – yet not necessarily *uniquely* – accepting automata: a word $x \in \Sigma^*$ has exact non-deterministic automatic complexity $k \in \mathbb{N}$ if there exists a non-deterministic automaton of k states which accepts x while rejecting every other word of the same length as x, and no automaton of fewer states has this property. Importantly, and in contrast to the classical notion, the witnessing automaton may have multiple paths of computation accepting x. We denote this measure of complexity by A_{Ne}, and study a class of languages of low A_{Ne}-complexity defined as $L_q = \{ x \in \Sigma^* \colon A_{Ne}(x) < q|x| \}$, which is parameterised by rationals $q \in (0, 1/2)$ (generalising a class of sets first studied by Kjos-Hanssen). We show that for every $q \in (0, 1/2)$, this class is neither context-free nor recognisable by certain Boolean circuits. In the process, we answer an open question of Kjos-Hanssen quantifying the complexity of $L_{1/3}$ in terms of Boolean circuits, and also prove the Shannon effect for A_{Ne}.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Theory of computation → Regular languages

Keywords and phrases Automatic complexity, automata theory, formal languages, Boolean circuits, Shannon effect

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.24

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2510.07696 [4]

Funding *Bjørn Kjos-Hanssen*: This work was partially supported by a grant from the Simons Foundation (#704836 to Bjørn Kjos-Hanssen).
Linus Richter: This work was fully supported by Singapore Ministry of Education grant MOE-000538-01.
Frank Stephan: This work was partially supported by Singapore Ministry of Education grant MOE-000538-01.

Acknowledgements Parts of this work have appeared in the first author's Bachelor's thesis submitted to the National University of Singapore.

[1] Corresponding author

© Joey Chen, Bjørn Kjos-Hanssen, Ivan Koswara, Linus Richter, and Frank Stephan;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Automatic complexity is a notion of complexity of finite words (strings) determined by witnessing automata, first introduced by Shallit and Wang in [35] as a Turing computable alternative to Kolmogorov complexity. It is an analogue for finite automata of Sipser's distinguishing complexity [37]. Classically, the automatic complexity of a word x over a finite alphabet Σ refers to the cardinality – counted in number of states[2] – of the smallest deterministic finite automaton which accepts x and rejects every other word of the same length as x [35]. The notion as well as variations of it have proven interesting for multiple reasons. For instance, since automatic complexity is Turing computable, it can be used in the study of computational complexity: the computational complexity of sets of binary words of low automatic complexity has helped prove missing relationships in the Complexity Zoo [1] (see [20, Theorem 39] for an example). Further, the detailed investigation of words in terms of their automatic complexity [17, 16] has shed light on *computable* notions of randomness, which are unavailable from the viewpoint of Kolmogorov complexity [21, 40, 28].

In this paper, we study a weakening of a variation of automatic complexity due to Hyde [12], and show that it generates classes of words too complicated to be captured by pushdown automata, nor by certain classes of constant-depth Boolean circuits – both of which are notably computationally more powerful than finite automata. This provides further evidence towards the conjecture that automatic complexity is hard to compute (see e.g. [15]).

1.1 Technical Background

Fix a finite alphabet Σ of at least two elements. In usual Kleene notation, we denote by Σ^* the set of all finite words of elements from Σ. We denote the empty string by ε, and the set of non-empty words by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. By an **automaton** we always mean a non-deterministic finite automaton, unless otherwise stated. We do not allow ε-transitions.

▶ **Definition 1.** *Let $x \in \Sigma^*$. An automaton M* **exactly accepts** *x if M accepts x, and whenever both $y \neq x$ and $|y| = |x|$ then M rejects y.*

The pumping lemma shows that this definition is maximally restrictive on the number of words accepted by the witnessing automaton; trying to strengthen the definition by asking for outright uniqueness of the accepted word only leads to trivialities.

▶ **Definition 2.** *The* **automatic complexity** *of $x \in \Sigma^*$ is given by*

$$A_D(x) = \min\{\, k \in \mathbb{N} \colon \text{there exists a DFA of } k \text{ states which exactly accepts } x \,\}.$$

For a reference on contemporary automatic complexity, see e.g. the recent [19]. The subscript D stands for "deterministic", indicating that $A_D(x)$ is determined by the smallest DFA. By definition, it is clear that A_D is well-defined, and even computable (for every $n \in \mathbb{N}$, there are only finitely many DFAs, and each can be simulated in finite time). However – similar to the unnatural properties of *plain* compared to *prefix-free* Kolmogorov complexity – the measure A_D has the following properties, which may render it undesirable as a natural measure of complexity of words. These were first described in [13]:

[2] A version of automatic complexity counting the number of *transitions* has been studied by Serna [32]; see also Shallit and Breitbart [34].

1. A_D is not invariant under natural transformations on strings, such as reversals. For instance, Hyde and Kjos-Hanssen have verified computationally that $A_D(011100) = 4 < 5 = A_D(001110)$.
2. The DFA witnessing $A_D(x)$ often appears unnatural, in the sense that determinism requires $A_D(x)$ to be total: in many cases, an automaton non-"deterministically" witnessing $A_D(x)$ needs to be augmented by an extra state to which every non-accepting path leads.

To overcome these obstacles, Hyde introduced automatic complexity witnessed by the smallest *non-deterministic* finite automaton (NFA) [12].

▶ **Definition 3.** *Let $x \in \Sigma^*$. An automaton M* **uniquely accepts** *x if M exactly accepts x and there is only one path in M which accepts x.*

Clearly, every DFA which exactly accepts x also uniquely accepts x. For NFAs, however, this is not the case. An NFA uniquely accepts x if and only if the NFA exactly accepts x and the NFA is unambiguous on $\Sigma^{|x|}$. Though Hyde [12] required the NFA to be unambiguous on $\Sigma^{|x|}$, she noted that the complexity based on NFAs is much more flexible and many words have a smaller complexity in her version than if only DFAs are considered. This led her to:

▶ **Definition 4.** *Let $x \in \Sigma^*$. The* **unique non-deterministic automatic complexity** *of x is given by*

$$A_N(x) = \min\{\, k \in \mathbb{N} : \text{there exists an NFA of } k \text{ states which uniquely accepts } x \,\}.$$

▶ Remark 5. We note that this notion is usually called "non-deterministic automatic complexity". As we study an ostensibly weaker notion below, we emphasise the additional strength of the notion defined in Definition 4 by adding the attribute "unique".

While it is well-known that NFAs and DFAs recognise exactly the same class of languages – the regular languages (see e.g. [33, 14] for a comprehensive background on automata theory) – the respective notions of automatic complexity differ. The following properties of A_N have been derived by Hyde and Kjos-Hanssen alongside co-authors, and others. Let $M_N(x)$ denote both the **minimal automaton witnessing** $A_N(x)$ and the directed graph representing it.

▶ **Lemma 6.** *Let $x \in \Sigma^*$.*
1. *$A_N(x) \leq (|x|/2) + 1$ follows from exhibiting suitable NFAs* [12].
2. *$M_N(x)$ is planar* [2].

Building upon Hyde's work from [12], in the present paper we study more closely the notion of automatic complexity induced by a weaker class of machines: the class of exactly but not necessarily uniquely accepting automata.

▶ **Definition 7.** *Let $x \in \Sigma^*$. The* **non-deterministic automatic complexity** *of x is*

$$A_{Ne}(x) = \min\{\, k \in \mathbb{N} : \text{there exists an NFA of } k \text{ states which exactly accepts } x \,\}.$$

Since every NFA which uniquely accepts x also exactly accepts x, we have $A_{Ne}(x) \leq A_N(x)$. Whether equality holds is still open (Question 48). In [20], Kjos-Hanssen investigated the complexity of certain languages induced by A_N in terms of more complicated models of computation, e.g. pushdown automata. In particular, he showed:

▶ **Theorem 8.**
1. *$\{\, x \in \{0,1,2\}^* : A_N(x) \leq |x|/2 \,\}$ is not context-free.*
2. *$\{\, x \in \{0,1\}^* : A_N(x) \leq |x|/3 \,\}$ cannot be recognised by constant-depth circuits with semi-unbounded fan-in, using Boolean $\wedge$- and $\vee$-gates.*

Results of this type motivate this paper: we investigate the impact of exactness on the behaviour of automatic complexity, which we describe via theorems akin to Theorem 8.

1.2 Our Theorems and the Structure of This Paper

We investigate the complexity of A_{Ne} as a function in terms of the complexity of the language of A_{Ne}-complicated words. Explicitly, we investigate the following class of languages first defined[3] by Kjos-Hanssen [20], and prove results on their complexities.

▶ **Definition 9.** *For $q \in (0, 1/2)$, define $L_q = \{ x \in \Sigma^* : A_{Ne}(x) < q|x| \}$.*

In Section 2, we isolate complexity results on the L_q-sets which follow from a fine-grained investigation of its elements. For instance, in Proposition 17 we isolate an upper bound of the Kolmogorov complexity of words in L_q. This gives a small-to-large result – a theorem about elements which provides information about sets – in the form of Corollary 19, which shows that the cardinality of $L_q \cap \Sigma^n$ is in $o(|\Sigma|^n)$. This observation also yields a proof of the **Shannon effect** for A_{Ne}:

▶ **Theorem 22.** *Let $A_{Ne}(\Sigma^n) = \max_{x \in \Sigma^n} A_{Ne}(x)$. For almost every $x \in \Sigma^*$ we have*

$$A_{Ne}(x) \geq A_{Ne}\left(\Sigma^{|x|}\right) - o\left(A_{Ne}\left(\Sigma^{|x|}\right)\right).$$

In Section 3, we show that pushdown automata are not powerful enough to characterise A_{Ne}-complicated words, which the following theorems show.

▶ **Theorem 34.** *For every $q \in (0, 1/2)$, the language L_q is not context-free.*

▶ **Theorem 35.** *For every $q \in (0, 1/2)$, the language $\Sigma^* \setminus L_q$ is not context-free.*

In Section 4, we consider the complexity of L_q in terms of Boolean circuits. To do so, we use two classical types of Boolean circuits – $\mathbf{SAC}^0$, defined in Section 4.1, and $\bigoplus\mathbf{SAC}^0$, defined in Section 4.2 – and apply a counting argument to prove:

▶ **Theorem 39.** *Let $q \in (0, 1/2)$ and $|\Sigma| = 2$. Then $L_q \notin \mathbf{SAC}^0$ and $\Sigma^* \setminus L_q \notin \mathbf{SAC}^0$.*

▶ **Theorem 46.** *Let $q \in (0, 1/2)$ and $|\Sigma| = p$ for some prime p. Then $L_q \notin \bigoplus\mathbf{SAC}^0$ and $\Sigma^* \setminus L_q \notin \bigoplus\mathbf{SAC}^0$.*

As a special case, we show that $L_{1/3}$ is not $\bigoplus\mathbf{SAC}^0$-recognisable, answering a question of Kjos-Hanssen [20, p. 351]. Omitted proofs as well as a partial generalisation to alphabets of non-prime cardinality can be found in the related full version of this paper [4].

In Section 5, we conclude this paper by giving a few open questions.

2 Combinatorial Properties of L_q

In this section, we derive combinatorial properties of L_q which are needed in the sequel, particularly to prove Theorem 34. Fix $q \in (0, 1/2)$. Firstly, we show that L_q satisfies a strong closure property: any word $x \in \Sigma^*$ can be extended to some word $y \in \Sigma^*$ for which $y \in L_q$.

▶ **Proposition 10.** *Suppose $x \in \Sigma^*$. If $m > |x|/q$ then $x^m \in L_q$.*

Proof. Let $n = |x|$ and suppose $x = x_0 \cdots x_{n-1} \in \Sigma^*$. Now build an NFA as follows: there are n states $\{s_0, \ldots, s_{n-1}\}$, with s_0 being both the start and unique accepting state. Transitions are given by $s_i \xrightarrow{x_i} s_{i+1}$ for $i < n-1$ and $s_{n-1} \xrightarrow{x_{n-1}} s_0$. It is readily seen that this automaton witnesses $A_{Ne}(x^m) \leq |x| < qm < qm|x| = q|x^m|$, as needed. ◀

[3] In [20, Def. 17], Kjos-Hanssen has considered the complementary decision problem, given by $q|x| < A_{Ne}(x)$. We note that our class $\{ L_q : q \in (0, 1/2) \}$ is more general.

While the previous proposition employs repetition of words to push the non-deterministic automatic complexity down, in the following lemma we show that spacing out bits of information achieves the same effect. W.l.o.g., assume $0 \in \Sigma$. For notation, if $x = x_0 \cdots x_{n-1} \in \Sigma^*$ then define the **(Hamming) weight of** x by $\text{weight}(x) = |\{\, k < n \colon x_k \neq 0 \,\}|$.

▶ **Lemma 11** (Gap Lemma). *For every $c \in \mathbb{N}$ there exists $n \in \mathbb{N}$ such that if $x \in \Sigma^n$ and $\text{weight}(x) \leq c$ then $x \in L_q$.*

Note that, in the statement above, n depends on q, which we fixed at the beginning of this section. Before we give the proof, we need the following number-theoretical lemma, called Bertrand's postulate (for a proof see e.g. [29]). Let $\mathbb{P}$ denote the set of prime numbers.

▶ **Lemma 12** (Bertrand's postulate). *If $h > 1$ then $\mathbb{P} \cap (h, 2h)$ is non-empty.*

Proof of Proposition 11. Fix $c \in \mathbb{N}$. For each $n \in \mathbb{N} \setminus \{0, 1\}$, we define a finite sequence of primes by $(p_1(n), \ldots, p_c(n))$ as follows: put $p_1(n) = \min\left(\mathbb{P} \cap (\sqrt[c]{n}, 2\sqrt[c]{n})\right)$ and

$$p_{i+1}(n) = \min(\mathbb{P} \cap (p_i, 2p_i)) \qquad \text{for } i = 1, 2, \ldots, c-1.$$

Since $n > 1$, Bertrand's Postulate shows that this is well-defined. Now, let

$$Q_i(n) = \left(\frac{1}{p_i(n)}\right) \prod_{j=1}^{c} p_j(n)$$

Bertrand's postulate alongside a short calculation implies $p_c(n) < 2^{c-1} p_1(n) < 2^c \sqrt[c]{n}$, and so

$$Q_i(n) \leq (p_c(n))^{c-1} \leq 2^{c(c-1)} n^{\frac{c-1}{c}}$$

which proves that $Q_i(n) \in O\left(n^{\frac{c-1}{c}}\right)$. This also shows that, in the limit, $Q_i(n) < n$. Similarly, $Q_i(n)p_i(n) > (p_1(n))^c > (\sqrt[c]{n})^c = n$ and hence, again in the limit, $Q_i(n) < n < Q_i(n)p_i(n)$. For $x \in \Sigma^n$ with $\text{weight}(x) \leq c$, write $x = w_0 0^{\ell_1} w_1 \cdots 0^{\ell_k} w_k$ for some $k \leq c$. Given our fixed $q \in (0, 1/2)$, we now choose $n \in \mathbb{N}$ sufficiently large so that we may assume the following (write $Q_i = Q_i(n)$):

- $|w_i| \leq cQ_1$ for $i = 0, 1, \ldots, k$.
- $\ell_i \geq Q_1$ for $i = 1, \ldots, k$.

Now, write $\ell_i = a_i Q_i + r_i$ where $0 \leq r_i < Q_i$. Since $Q_i p_i > n$ we must have $a_i < p_i$; otherwise, $|\ell_i| > n = |x|$, a contradiction. Hence, consider the automaton M in Figure 1.

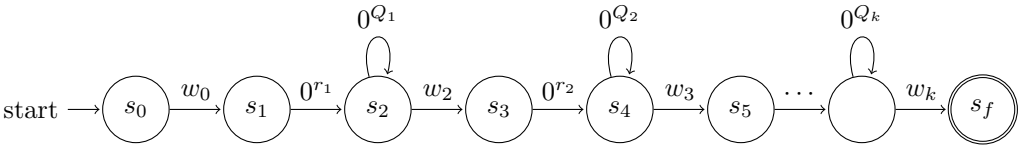


Figure 1 The non-deterministic automaton witnessing the "gap lemma".

We show that M is as required. First, by definition, M accepts x. To show exactness, suppose $y \in \Sigma^n$ and that M accepts y. If $x \neq y$, assume w.l.o.g. that $M(y)$ traverses the 0^{Q_1}-loop fewer than a_i-many times. Since $|y| = n$, $M(y)$ must go through the remaining loops more often to make up for the Q_1-deficit. However, the equation $Q_1 = d_2 Q_2 + \ldots + d_k Q_k$ has no integer solution, since p_1 divides the right-hand side yet not Q_1. Thus, M cannot accept y, as needed. Finally, recall that $r_i, |w_i| \leq cQ_1 \in O\left(n^{\frac{c-1}{c}}\right)$. Thus, for n large enough, inspection of the automaton M shows that $A_{Ne}(x) \leq 3kcQ_1 \in O\left(n^{\frac{c-1}{c}}\right)$ for every $x \in \Sigma^n$, and so $A_{Ne}(x) < q|x|$. ◀

As a consequence of our proof, we also obtain the following (we thank the anonymous reviewer for pointing this out).

▶ **Corollary 13.** *For every $c \in \mathbb{N}$ and every $q \in (0, 1/2)$ there exists $n \in \mathbb{N}$ such that if $x \in \Sigma^{>n}$ and* $\mathrm{weight}(x) \leq c$ *then $x \in L_q$.*

Our next result studies the small-scale structure of words in L_q. We say w is a **factor of** x if there exist $u, v \in \Sigma^*$ for which $x = uwv$; we write $w \preceq x$. If $u \in \Sigma^+$ or $v \in \Sigma^+$ then w is a **proper factor of** x; we write $w \prec x$. Call a non-empty word w a **square** if there exists $v \prec w$ for which $w = vv$; we write $w = v^2$.

▶ **Proposition 14.** *Let $n \geq 4$ and $x \in L_q \cap \Sigma^n$. There exists a proper factor $w \prec x$ of length $|w| \geq \left(\frac{1-2q}{2}\right)\sqrt{n}$ for which there are $u, v \in \Sigma^+$ with $|u| = |v| \leq |w|$ and $uw = wv \prec x$. Further, $uwv \prec x$.*

Note that if $|u| = |v| = |w|$ then the conclusion of Proposition 14 yields a square. To prove the general case of Proposition 14, we again need a classical auxiliary result, in this case due to Lyndon and Schützenberger [27].

▶ **Theorem 15** (The First Lyndon-Schützenberger-Theorem)**.** *Suppose $x, y \in \Sigma^*$. Then $xy = yx$ if and only if there exists $z \in \Sigma^*$ and $k, \ell \in \mathbb{N}$ for which $x = z^k$ and $y = z^\ell$.*

Note that the First Lyndon-Schützenberger-Theorem characterises *bordered words*[4] – those which have a non-trivial decomposition of the form $uw = wv$ – as those generated by powers of a common word z. This will be important in the proof of Proposition 14. We also require the following combinatorial lemma.

▶ **Lemma 16.** *Suppose $x \in \Sigma^n$ for some $n \geq 4$. Assume $x \in L_q$, and let $M_{Ne}(x)$ be the witnessing automaton with accepting run $(q_0, \ldots, q_n)$. Then*

$$|\{\, k \in \mathbb{N} \colon (\exists\, i, j)(i < j < k \wedge q_i = q_j = q_k)\,\}| \geq (1 - 2q)n.$$

Proof. Consider the list of states $(q_0, \ldots, q_n)$. Since $q < 1/2$, we have $2qn < n$. In particular, $n = 2qn + (1-2q)n$. Hence, by the pigeonhole principle, there exist at least $(1-2q)n$ indices at which some state is visited a third time. ◀

We now prove Proposition 14. Call triples (i, j, k) as provided by Lemma 16 **loop triples (for x)**. Before we give the proof of Proposition 14, we introduce the following notation: write $x_{[i,j]} = x_i \cdots x_j$. For instance, if $n \geq 4$, then $x_0 x_1 \cdots x_{n-1} = x_{[0,n-1]} = x_{[0,2]} x_{[3,n-1]}$.

Proof of Proposition 14. Let $x \in \Sigma^n$ be as assumed, and suppose $(q_0, \ldots, q_n)$ is the run of M_{Ne} which accepts x. Observe that if (i, j, k) is a loop triple for x (by Lemma 16 there are at least $(1 - 2q)n$ many), then the witnessing NFA $M_{Ne}(x)$ has completed at least two loops by the time it has read the word $x_{[0,k-1]}$. There are two cases.

1. There exists a loop triple (i, j, k) for which $\max\left(\left|x_{[i,j-1]}\right|, \left|x_{[j,k-1]}\right|\right) > (1 - 2q)\sqrt{n}$. Assume w.l.o.g. that $\left|x_{[j,k-1]}\right| \geq \left|x_{[i,j-1]}\right|$ and write $x = x_{[0,i-1]} x_{[i,j-1]} x_{[j,k-1]} x_{[k,n-1]}$. Since (i, j, k) is a loop triple, $q_i = q_j = q_k$, and thus $M_{Ne}(x)$ also accepts the word $x_{[0,i-1]} x_{[j,k-1]} x_{[i,j-1]} x_{[k,n-1]}$. Since $M_{Ne}(x)$ exactly accepts x, we have $x_{[j,k-1]} x_{[i,j-1]} =$

[4] For more on bordered words, see e.g. [30]. A more general characterisation is given by the Second Lyndon-Schützenberger-Theorem 18.

$x_{[i,j-1]}x_{[j,k-1]}$, and so Theorem 15 implies $x_{[i,j-1]} = z^k$ and $x_{[j,k-1]} = z^\ell$ for some $z \in \Sigma^+$ and $k, \ell \in \mathbb{N}$. Since $k, \ell \geq 1$, the decomposition of $x_{[i,k-1]}$ trivialises into a product of copies of z:

$$x_{[i,k-1]} = x_{[i,j-1]}x_{[j,k-1]} = zz^{k+\ell-1} = z^{k+\ell-1}z$$

As $\left|z^{k+\ell-1}\right| \geq \left|x_{[j,k-1]}\right| \geq (1-2q)n > (1-2q)\sqrt{n}$, we are done.

2. For all loop triples (i,j,k) we have $\max\left(\left|x_{[i,j-1]}\right|, \left|x_{[j,k-1]}\right|\right) \leq (1-2q)\sqrt{n}$.
By Lemma 16, there exist $(1-2q)n$ indices k for which there exist (i,j) such that (i,j,k) is a loop triple. Since every loop in a loop triple has length at most $(1-2q)\sqrt{n}$, the pigeonhole principle gives an $\ell \leq (1-2q)\sqrt{n}$ such that there exist at least $m \geq \sqrt{n}$ such indices k at which a loop of length ℓ was just completed (hence, we only focus on the *second* loops in each loop triple). Let this set of indices be given in ascending order, denoted by $\mathcal{K} = \{k_1, \ldots, k_m\}$, with associated loops $\rho_1, \ldots, \rho_m \prec x$, each of length ℓ.
We show that ρ_1 and ρ_m must be disjoint, i.e. share no states along their traversals in $M_{Ne}(x)$. Let q_{k_1} be the origin state of the loop ρ_1. By definition, ρ_1 is the second loop in the loop triple (i_1, j_1, k_1). Suppose τ is the first loop at q_{k_1} so that $\tau\rho_1$ is a loop triple at q_{k_1}. Then, if we read $b > (1-2q)\sqrt{n}$ letters along the loops at state q_{k_1}, then we could concatenate those loops with τ to obtain a loop triple, one of whose lengths exceeds $(1-2q)\sqrt{n}$, which contradicts the assumption of this case. Therefore, at state q_{k_1}, we can only read at most $(1-2q)\sqrt{n}$ letters of the factors contained in $\rho_1, \ldots, \rho_m$, before moving on to a different state, never to return. However, by construction, for every $i \leq m$ we know that x_{k_i} appears in ρ_i, and thus we must read at least $m \geq \sqrt{n}$ letters throughout all loops $\rho_1, \ldots, \rho_m$. Since $q < 1/2$, we have $(1-2q)\sqrt{n} < \sqrt{n} \leq m$; hence, the first and last loops ρ_1 and ρ_m must be disjoint. Thus, $x = u\ \rho_1\ y\ \rho_m\ u'$ where $u, y, u' \prec x$ and $|\rho_1| = |\rho_m| = \ell$. By exact acceptance of $M_{Ne}(x)$, we have

$$x = u\ (\rho_1)^2\ y\ u'$$

since $|\rho_1| = |\rho_m|$. Therefore, $\rho_1 y = y\rho_m$, and thus, with $y' = \rho_1 y$, we have $y'\rho_m = \rho_1 y'$. To show that y' has the desired length, note that $y\rho_m$ must contain the set $\{x_{k_2}, \ldots, x_{k_m}\}$; the loop ρ_1, since it is the first loop in $\mathcal{K}$, can only contain x_{k_1}. Since $n \geq 4$, we have

$$|y'| = |y\rho_m| \geq |\mathcal{K}| - 1 = m - 1 \geq \sqrt{n} - 2 \geq \frac{\sqrt{n}}{2}.$$ ◀

We now apply Proposition 14 to go even finer: instead of studying the complexity of L_q, we classify the complexity of *words* in L_q, using plain Kolmogorov complexity. Fix an alphabet Σ of cardinality k, and let C_k denote the **plain Kolmogorov complexity** on words in Σ:

$$C_k(x) = \min\{\, \ell(p) \colon U_k(p) = x \,\}.$$

where U_k is a universal Turing machine on the k-element alphabet. For details on Kolmogorov complexity, see e.g. [5].

▶ **Proposition 17.** *If $x \in \Sigma^n \cap L_q$, then*

$$C_k(x) \leq n - \frac{(1-2q)}{2}\sqrt{n} + 5\log_k(n) + O(1).$$

Its proof – which is included in the full version [4] – requires an extension of Theorem 15, which gives a sufficient and necessary criterion for the decomposition of words with same prefix and suffix. As it will be useful to us in the sequel outside of the proof of Proposition 17, we state it right here in the version of [33].

▶ **Theorem 18** (The Second Lyndon-Schützenberger-Theorem). *Let $x, y, z \in \Sigma^*$. Then $xy = yz$ iff there exist $e \in \mathbb{N} \setminus \{0\}$, $u \in \Sigma^+$ and $v \in \Sigma^*$ such that $x = uv$, $z = vu$, and $y = x^e u = u z^e$.*

With $|\Sigma| = k$ as before, note that the function which maps $x \in \Sigma^*$ to its C_k-witness is an injection. Hence, Proposition 17 immediately yields the following bound on $|L_q|$.

▶ **Corollary 19.** *The set $L_q \cap \Sigma^n$ has cardinality in $o(|\Sigma|^n)$.*

From Corollary 19, we now deduce the Shannon effect for A_{Ne}. Originally conjectured by Shannon [36] and proven (and named) by Lupanov for Boolean functions [25, 26], the Shannon effect says that most strings are of almost maximal complexity. We give a definition due to Wegener [45].

▶ **Definition 20.** *Let $P \subset \Sigma^*$. We say that* **almost all** x **have property** P *if*

$$\lim_{n\to\infty} \frac{|P \cap \Sigma^n|}{|\Sigma|^n} = 1.$$

▶ **Definition 21.** *Let Γ be a complexity measure defined on Σ^*. For $n \in \mathbb{N}$, let $\Gamma(\Sigma^n) = \max_{x\in\Sigma^n}(\Gamma(x))$. Then Γ has the* **Shannon effect** *if for almost all $x \in \Sigma^*$ we have*

$$\Gamma(x) \geq \Gamma\left(\Sigma^{|x|}\right) - o\left(\Gamma\left(\Sigma^{|x|}\right)\right).$$

By exhibiting upper and lower bounds of complexity for *all* words, it is readily seen that (plain and prefix-free) Kolmogorov complexity satisfy the Shannon effect [21, 41, 42, 23, 22], as do A_D [35] and A_n [12, 18]. The cardinality argument of Corollary 19 shows:

▶ **Theorem 22.** *A_{Ne} satisfies the Shannon effect.*

Proof. Fix $q = 1/(2+\epsilon)$ for some small $\epsilon > 0$. Since $A_{Ne}(x) \leq A_N(x) \leq (|x|/2) + 1$ (Lemma 6), identifying a suitable lower bound suffices. By Corollary 19, for $o(|\Sigma|^n)$-many words $x \in \Sigma^n$ we have $x \in L_q$. Hence, for almost all (as per Definition 20) $x \in \Sigma^n$,

$$\frac{n}{2+\epsilon} \leq A_{Ne}(x) \leq \frac{n}{2} + 1$$

and so $A_{Ne}(x)/|x|$ is sufficiently close to $1/2$ for typical large x. ◀

3 L_q Is Not Context-Free

Fix $q \in (0, 1/2)$ and suppose w.l.o.g. that $0, 1 \in \Sigma$. In this section, we demonstrate that L_q cannot be generated by a context-free grammar (CFG); hence, L_q is not context-free. To this end, we first define the concept of a *rich* CFG. We then prove that if a CFG generates L_q, it must be rich. Finally, we show that any rich CFG generates words of arbitrarily high complexity, which contradicts the fact that the CFG generates L_q.

We provide the required definitions. (For more details, see e.g. [33].) A **context-free grammar** (CFG) is a tuple $G = (V_T, V_N, S, P)$ where:

- V_T is the set of **terminal symbols**.
- V_N is the set of **non-terminal symbols**.
- $S \in V_N$ is the **start symbol**.
- P is a finite set of **productions**.

We also insist that $V_T \cap V_N = \emptyset$, and we define the set of **symbols** by $V = V_T \cup V_N$. Elements of V^* are called **sentential forms**. Productions in P are pairs (A, γ) where $A \in V_N$ and $\alpha \in V^*$. We denote such a production by $A \to \gamma$.

The **derivation relation** $\Longrightarrow$ is defined as follows: if $\alpha, \beta \in V^*$ then $\alpha \Longrightarrow \beta$ if $\alpha = \alpha' A \alpha''$ and $\beta = \alpha' \gamma \alpha''$ for some $\alpha', \alpha'' \in V^*$, and there exists a production $A \to \gamma$ in P. The transitive and reflexive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

A language that is recognisable by a CFG is called a **context-free language** (CFL).

▶ **Definition 23.** *A CFG has* **no useless nonterminals** *if:*
1. *each nonterminal is reachable from the starting symbol; and*
2. *a terminal string can be derived from each nonterminal.*

▶ **Definition 24.** *Let G be a CFG. A nonterminal $A \in G$ is a* **rich nonterminal** *if for some words $v, w, x, y \in \Sigma^*$ we have $vwxy \neq \varepsilon$ and $A \Longrightarrow^* vAx \mid wAy$ as well as:*
1. *if $vw \neq \varepsilon$ then $vw \neq wv$; and*
2. *if $xy \neq \varepsilon$ then $xy \neq yx$.*

A **rich CFG** *has a rich nonterminal but no useless nonterminals. A* **rich CFL** *is generated by a rich CFG.*

Our motivation for rich CFGs follows from Theorem 15, however, we note here that, in style, our richness characterisation is similar to classical results by Ginsburg [9, Theorem 5.5.1], who characterised boundedness of CFLs via syntactical properties of grammars. Our syntactical notion of richness, similarly, characterises the complexity of generated languages, in our case L_q. The equivalence in Theorem 15 implies that a rich non-terminal can construct words which do not collapse to repeating copies of a common factor z. This is needed in Section 3.2, where we construct high-complexity words.

3.1 Only Rich CFGs Can Generate L_q

We require the following normal form theorem due to Greibach [10] (see [11, p. 277] for a modern exposition).

▶ **Theorem 25** (Greibach Normal Form Theorem)**.** *Every CFG with no ε-productions can be expressed in* **Greibach Normal Form***: all its production rules are of the form $A \to x\overline{A}$ where $x \in \Sigma$ and $\overline{A}$ is a finite word of nonterminals.*

The following corollary is immediate.

▶ **Corollary 26.** *Every CFL omitting ε is generated by a CFG in Greibach Normal Form.*

Our main result in this subsection is the following.

▶ **Theorem 27.** *If L_q is generated by a context-free grammar G, then G is rich.*

Proof. By our results in the previous section, L_q is non-empty; further, by definition, $\varepsilon \notin L_q$. So, by Corollary 26, there exists a CFG G in Greibach Normal Form which generates L_q. We show that G must be rich by a counting argument on the number of nonterminals of G. Let $k \in \mathbb{N}$ denote the number of nonterminals in G. Define

$$x_i = 0^i 1^{4k-i} \qquad \text{for } i = 1, 2, \ldots, 4k-1.$$

By Proposition 10, for every i there exists $m_i \in \mathbb{N}$ for which $x_i^{m_i} \in L_q$. Similarly, for each i there exists $m_i' \in \mathbb{N}$ for which the derivation tree of $x_i^{m_i'}$ has a branch which contains some nonterminal A at least $(4k)^2 + 1$ times. Let $M \in \mathbb{N}$ be sufficiently large to satisfy these

requirements for all x_i simultaneously. By the pigeonhole principle, there exist $i, j, \ell \leq 4k-1$ such that some nonterminal A appears at least $(4k)^2 + 1$ times in some branch of the derivation tree of each of x_i^M, x_j^M and x_ℓ^M.

Consider such a sufficiently long branch of the derivation tree of x_i^M, in which we choose to expand A at the end. Since G is in Greibach Normal Form, the derivation is of the form

$$S \Longrightarrow^* y_i^1 y_i^2 y_i^3 \dots y_i^s A z_i^s \dots z_i^3 z_i^2 z_i^1$$

from which x_i^M can be derived in at least $(4k)^2+1$ expansions of A. Observe that each $y_i^j \neq \varepsilon$, since G is in Greibach Normal Form. Consider the number of expansions of A in terms of blocks $B_1, B_2, \dots, B_n$ such that each block has cardinality $4k$. By assumption, $n \geq 4k+1$. Let A_m be the derivation of A from the expansions in block B_m. There are two cases:

1. For some $m \leq n$, $A_m = yA$ with $y \in \Sigma^*$ and[5] $|y| \geq 4k$.
2. For all $m \leq n$, $A_m = y_m A z_m$ with $y_m, z_m \in \Sigma^+$. Then, $A_{4k} = yAz$ with $|y|, |z| \geq 4k$.

Since these two cases apply to all x_i^M, x_j^M and x_ℓ^M, two of them must share the same case above. W.l.o.g. assume both x_i^M and x_j^M fall into case 2 (the argument for case 1 is similar). Hence, $T \Longrightarrow^* yAz$ (from the derivation of x_i^M) and $T \Longrightarrow^* vAw$ (from the derivation of x_j^M) where $|y|, |z|, |v|, |w| \geq 4k$. By definition, y, z contain i zeroes, while v, w contain j zeroes among the first $4k$ letters. It is now seen from the First Lyndon-Schützenberger-Theorem that $yv \neq vy$ and $zw \neq wz$. Hence, A is a rich nonterminal. ◀

3.2 Every Rich CFG Generates High-Complexity Words

In this section, we prove that every rich CFG generates words of arbitrarily high complexity relative to its length. In particular, there exists a word x for which $A_{Ne}(x) > q|x|$ for every $q \in (0, 1/2)$. This contradicts the fact that any rich CFG can generate L_q for any q, since any $x \in L_q$ satisfies $A_{Ne}(x) < q|x|$. We also isolate the following technical proposition.

▶ **Proposition 28.** *Suppose $u, v \in \Sigma^n$ with $uv \neq vu$. Then the following set is infinite:*

$$\mathcal{I}_{(u,v)} = \{\, x \in \{u, v\}^* \colon \textit{if } y \prec x \textit{ satisfies } |y| > 2\log(|x|) \textit{ then } y \textit{ occurs exactly once in } x \,\}$$

Proving Proposition 28 takes a few technical lemmas on the behaviour of non-commuting strings in formal languages. Denote the **set of factors** of $w \in \Sigma^*$ by $[w] = \{\, x \in \Sigma^* \colon x \prec w \,\}$. For convenience, we now fix some $n \in \mathbb{N}$ and a pair $u, v \in \Sigma^n$ for which $uv \neq vu$.

▶ **Lemma 29.** $uv, vu \notin [u^3] \cup [v^3]$

Proof. We give the argument for $uv \notin [u^3]$; the other parts are similar. Assume that $uv \in [u^3]$; thus write $u^3 = xuvy$ for some $x, y \in \Sigma^*$. Note that $|xy| = |u| = |v|$. Since $uv \neq vu$ we cannot have $x, y \in \{u, v\}$, and thus $|x|, |y| < |u| = |v|$. But now, by periodicity of u^3, we must have $xy = u$. Thus $u^3 = xyxyxy = xuvy$. Therefore, $uv = yxyx$, from which it follows that $u = xy = yx = v$, contradicting the fact that $uv \neq vu$. ◀

To motivate the next lemma, we provide the definition of string homomorphisms.

▶ **Definition 30.** *A function $h \colon \{0, 1, \dots, n-1\}^* \to \Sigma^*$ is a* **string homomorphism** *if for all $n_i \in \{0, 1, \dots, n-1\}$ we have $h(n_0 \cdots n_k) = h(n_0) \cdots h(n_k)$.*

[5] This is a consequence of the observation immediately following Theorem 25.

Observe that every such string homomorphism is uniquely defined by its action on the alphabet. Define a string homomorphism $h\colon \{0,1,2\} \to \{u,v\}^*$ given by

$$h(0) = uv \qquad h(1) = vu \qquad h(2) = u^3v^4$$

With this string homomorphism fixed, the following lemma is immediate from Lemma 29.

▶ **Lemma 31.** $u^4, v^4 \notin \bigcup\{\, [x] \colon x \in h(\{0,1\}^*)\,\}$

To give a proof of Proposition 28, we first code words as follows. For every $k \in \mathbb{N}$, let σ_k be the lexicographical concatenation of all positive integers which, coded in binary, have length k; each is then followed by a 2. For instance, $\sigma_2 = 002012102112$. We consider the images of these words under h, and collect some immediate properties of the σ_k and the $h(\sigma_k)$ below, whose proofs are readily deduced, hence omitted.

▶ **Lemma 32.** *Let $k \in \mathbb{N}$.*

1. $|\sigma_k| = 2^k(k+1)$
2. $|h(\sigma_k)| = 2^k|v|(2k+7)$
3. $2\log(|h(\sigma_k)|) \geq 2k + 14|v|$ *for sufficiently large k.*

To prove Proposition 28, we show that for large enough k, every substring of $h(\sigma_k)$ of length at least $2\log|h(\sigma_k)|$ must contain two copies of $h(2)$; since the word between any two copies of $h(2)$ is unique within $h(\sigma_k)$, the proposition is proven.

Proof of Proposition 28. Fix $k \in \mathbb{N}$ sufficiently large so as to satisfy Item 3, and consider the word $h(\sigma_k)$. By construction and the choice of k, if $y \prec h(\sigma_k)$ and $|y| \geq 2\log(|h(\sigma_k)|)$ then y contains two copies of $h(2)$. By definition, $v^4 \prec h(2)$; on the other hand, Lemma 31 shows that v^4 cannot be a factor of any $h(w)$ with $w \in \{0,1\}^*$. Hence, $v^4 \prec y$ must be a factor of some $h(2)$ occurring in $h(\sigma_k)$. We show that the copies of $h(2)$ in y and in $h(\sigma_k)$ overlap perfectly. Consider the word $h(2)z = xh(2)$ contained in $h(\sigma_k)$. This bordered word is in fact a proper square, which can seen by a case analysis on x. Write

$$\begin{aligned} x &= a|u| + \ell && \text{for some } a \in \mathbb{N}, 0 \leq \ell < |u| \\ u &= \alpha\beta && \text{where } |\alpha| = \ell \\ v &= \gamma\delta && \text{where } |\gamma| = \ell \end{aligned}$$

and note that this renaming implies $|\beta| = |\delta|$, and that

$$h(2)z = xh(2) = x_1x_2(\alpha\beta)^3(\gamma\delta)^4 = (\alpha\beta)^3(\gamma\delta)^4z.$$

There are four cases describing x_1; using Theorems 15 and 18, each will lead to a contradiction.

1. $a = 0$: Since $|\alpha| = \ell$, this case implies $\alpha\beta = \beta\alpha$. Further, $|\beta\gamma| = |\alpha\beta| = |\beta\alpha|$, and so $\alpha = \gamma$. By comparing lengths, it is easily seen that $\beta = \delta$, and so $u = \alpha\beta = \gamma\delta = v$, a contradiction.
2. $a = 1$: Since $u = \alpha\beta$, by comparing initial segments it is readily seen that in this case $uv \prec v^4$, contradicting Lemma 29.
3. $a = 2$ or $a = 3$: Since $xh(2) = h(2)z$, we have $|x| = |z|$. So, if $a = 2, 3$, then again by comparing initial segments it is readily seen that $uv \prec v^4$, contradicting Lemma 29.
4. $a > 3$: Here, $v^4 \prec z$. Since $z = h(w)$ for some $w \in \{0,1\}^*$, Lemma 29 gives a contradiction.

Hence, the copies of $h(2)$ appearing in y are exactly those appearing in $h(\sigma_k)$. But now, if $y' \prec h(\sigma_k)$ is of length at least $2\log(|h(\sigma_k|)$, then it contains a factor of the form $h(2)\rho h(2)$ where $\rho \in h(\{u,v\}^*)$. Each such ρ appears only once in $h(\sigma_k)$, by construction. Thus, for large enough k, the word $h(\sigma_k)$ is as required, and thus the set $\mathcal{I}_{(u,v)}$ is infinite. ◀

▶ **Theorem 33.** *If G is a rich CFG, then G generates a word $x \in \Sigma^*$ such that $A_{Ne}(x) > q|x|$ for every $q \in (0, 1/2)$.*

For notation, if $\sigma \in \{0,1\}^*$, let $\overline{\sigma}$ denote the reverse of σ. Further, if $x, y \in \Sigma^*$ satisfy $xz = zy$ and both $xz, zy \prec w$ such that xz and zy overlap at z, then call xzy its **union**, written as $xz \cup zy$. We use Proposition 28.

Proof. Let G be a rich CFG with rich nonterminal A and witnesses $x, y, x', y' \in \Sigma^*$ for which $A \Longrightarrow^* xAy \mid x'Ay'$ and $xx' \neq x'x$ and $yy' \neq y'y$. Define $u_1 = xx', v_1 = x'x$ and $u_2 = yy', v_2 = y'y$. Now define string homomorphisms g, h by:

$$g(0) = u_1v_1 \qquad g(1) = v_1u_1 \qquad g(2) = u_1^3v_1^4$$
$$h(0) = u_2v_2 \qquad h(1) = v_2u_2 \qquad h(2) = u_2^3v_2^4$$

Now, fix any $w_1, w_2, w_3 \in \Sigma^*$ for which

$$S \Longrightarrow^* w_1Aw_3 \qquad \text{and} \qquad A \Longrightarrow^* w_2. \tag{$*$}$$

By repeated application of the generation rules in $(*)$, it is readily seen that for any $m, k \in \mathbb{N}$, the word $y_{m,k}$ of the following form is generated by G:

$$y_{m,k} = (w_1\ g(\sigma_k))\ (x^m\ w_2\ y^m)\ (\overline{h(\sigma_k)}\ w_3).$$

We show that, for sufficiently large m, k, the word $y_{m,k}$ has large non-deterministic automatic complexity. Choose m, k large enough so that $|y_{m,k}| \gg |w_1w_2w_3|$, and let $n = |y_{m,k}|$. Since we may choose k, m freely, we may also impose that

$$2\log(n) \leq m|x| \leq 3\log(n) \in o(\sqrt{n}). \tag{$\dagger$}$$

Now, let $z \prec y_{m,k}$ whose length is in $O(\sqrt{n})$ be the first occurrence of a word in $y_{m,k}$ of the form $zb = cz$ for words $b, c \prec y_{m,k}$. Below, we show that this is only possible if $b = c = \varepsilon$.

By choosing m, k wisely, we may assume that $|z|$ is even. Further, it will be convenient to distinguish the words which make up the left-hand and right-hand squares of z; hence, write $z = z_1z_2 = z_1'z_2'$ so that $|z_1| = |z_2|$ and $z_1 = z_1', z_2 = z_2'$, and $z_1z_2b = cz_1'z_2'$.

We show that $z_1 \prec w_1g(\sigma_k)$; the case that $z_2' \prec \overline{h(\sigma_k)}w_3$ is similar. Note that, otherwise, we may choose k large enough so that z_1 intersects $\overline{h(\sigma_k)}w_3$, and in particular, we may enforce that this intersection $s \in \Sigma^*$ has length at least $2\log(n)$. By construction and the fact that $z_1z_2b = cz_1'z_2'$, the word s must appear twice in $\overline{h(\sigma_k)}$, which contradicts Proposition 28.[6]

Thus, $z_1 \prec w_1g(\sigma_k)$ and $z_2' \prec \overline{h(\sigma_k)}w_3$ imply that $x^mw_2y^m$ is a factor of the union $z_2b \cup cz_1'$. By a counting argument, it is seen that either $x^m \prec z_2$ or $y^m \prec z_1'$. If $x^m \prec z_2$ – the other case is similar – then also $x^m \prec z_2' \prec \overline{h(\sigma_k)}$. But this is impossible by Proposition 28 and since $|z_2'| \geq m|x| \geq 2\log(n)$ by $(\dagger)$. Therefore, we have arrived at a contradiction: we can only have $z_1z_2b = cz_1'z_2'$ if $b = c = \varepsilon$. But now, the contrapositive of Proposition 14 shows that $y_{m,k} \notin L_q$ for every $q \in (0, 1/2)$. Since $y_{m,k}$ is generated by G, the result is proven. ◀

Theorem 33 and Theorem 27 combined imply our main result of this section:

▶ **Theorem 34.** *For every $q \in (0, 1/2)$, the language L_q is not context-free.*

[6] See in particular the proof of Proposition 28 to note that $\mathcal{I}_{(u_2,v_2)}$ can be generated by sets of the form $h(\sigma_k)$, and by a similar argument, by those of the form $\overline{h(\sigma_k)}$.

A language L is **CFL-immune** if it contains no infinite context-free language as a subset. We note here that L_q cannot be CFL-immune, since for every letter $x \in \Sigma$, the regular language $\{x\}^+$ is contained in L_q (modulo finitely many words, depending on q), and each of its words has constant complexity. However, the following holds:

▶ **Theorem 35.** *For every $q \in (0, 1/2)$, the language $\Sigma^* \setminus L_q$ is CFL-immune.*

For the proof, we direct the reader to [4]. Since $A_{Ne}(x) \leq A_D(x)$ for all words x, Theorem 35 also implies:

▶ **Corollary 36.** *For every $q \in (0, 1/2)$, $\{\, x \in \Sigma^* \colon A_D(x) \geq q|x| \,\}$ is CFL-immune.*

We conjecture that recognising L_q can be done via a linearly bounded automaton.

4 L_q Cannot Be Recognised by Certain Constant-Depth Circuits

In this section, we expand on our work in Section 3 by investigating the complexity of L_q further. Instead of considering pushdown automata, in this section we consider constant-depth circuits. We show that two types of circuits cannot recognise L_q either, which is analogous to Theorem 34 for pushdown automata.

Fix $q \in (0, 1/2)$ and fix $\Sigma = \{0, 1\}$. We first provide the definitions of two types of constant depth circuits explicitly – the class $\mathbf{SAC}^0$ in Section 4.1, and $\bigoplus\mathbf{SAC}^0$ in Section 4.2 – and then show that neither can recognise L_q, nor its complement.

4.1 The Circuit Class SAC^0

Suppose $k \geq 1$. A language L is **$\mathbf{SAC}^k$-recognisable** if it is recognised by a polynomial-size, $O(\log^k n)$-depth, uniform[7] semi-unbounded fan-in circuit (a circuit is **semi-unbounded** if it has unbounded fan-in-$\vee$, bounded fan-in-$\wedge$, and admits negative literals but no other negations [3]; cf. [44, 20]). Of these classes of particular interest is $\mathbf{SAC}^1$, since it is the class **logCFL** of languages which are log-space reducible to context-free languages [43, 44]. More generally, $\mathbf{SAC}^k$ enjoys the following relationship with the classical classes $\mathbf{AC}^k$ and $\mathbf{NC}^k$:

$$\mathbf{NC}^k \subseteq \mathbf{SAC}^k \subseteq \mathbf{AC}^k \subseteq \mathbf{NC}^{k+1} \qquad \text{for all } k \geq 1.$$

Just like $\mathbf{NC}^k$ and $\mathbf{AC}^k$, the class $\mathbf{SAC}^k$ is also closed under complements [3, Corollary 15].

Here, we consider the class $\mathbf{SAC}^0$. Contrary to the classes above, $\mathbf{SAC}^0$ is *not* closed under complementation [3]. Note that $\mathbf{SAC}^0$-circuits have *constant* depth; hence, the $\mathbf{SAC}^0$-recognisable languages can be characterised by formulas in a simple propositional language, as expressed in Lemma 38. We give a formal definition of $\mathbf{SAC}^0$ due to Kjos-Hanssen [20].

▶ **Definition 37.** *A language $L \subset \{0, 1\}^*$ is **$\mathbf{SAC}^0$-recognisable** if there exists a family $(C_i)_{i\in\mathbb{N}}$ of Boolean circuits which recognises L and which satisfies the following:*

1. *Each C_i is defined over the basic set $\{\wedge, \vee\}$ and accepts negative literals.*
2. *The family $(C_i)_{i\in\mathbb{N}}$ has constant depth.*
3. *Each C_i has unbounded fan-in-$\vee$ and bounded fan-in-$\wedge$.*
4. *Each C_i accepts words of length i.*

[7] Requiring uniformity is debatable; see e.g. [20, Remark 29].

Note that, for the classes $\mathbf{SAC}^k$ with $k > 0$, the size of the circuit must be polynomial in n. However, this requirement is redundant for $\mathbf{SAC}^0$ (cf. [20, Remark 30]). An important characterisation of $\mathbf{SAC}^0$-recognisable languages follows (cf. [20] and [3, p. 560]).

▶ **Lemma 38.** *A language $L \subset \Sigma^*$ is $\mathbf{SAC}^0$-recognisable iff there exists $c \in \mathbb{N}$ such that: for every $n \in \mathbb{N}$ and every $x \in \Sigma^n$ there exist $k_n \in \mathbb{N}$ and a Boolean formula $\psi_n = \bigvee_{i=1}^{k_n} \varphi_{i,n}$ for which $\varphi_{i,n}$ is a conjunction of at most c literals, and*

$$x \in L \iff \psi_n(x) \text{ holds.}$$

Using this lemma, which can be deduced from the distributive properties of propositional languages, we conclude:

▶ **Theorem 39.** *Let $q \in (0, 1/2)$ and $|\Sigma| = 2$. Then $L_q \notin \mathbf{SAC}^0$ and $\Sigma^* \setminus L_q \notin \mathbf{SAC}^0$.*

Proof. The proof uses a counting argument using Lemma 38. First, suppose $L_q \in \mathbf{SAC}^0$, witnessed by a sequence of formulas $(\psi_n)_{n\in\mathbb{N}}$ and a constant c. Consider ψ_n. Since $\varphi_{1,n}$ mentions at most c variables, the circuit accepts every word which agrees on these c variables. Hence, ψ_n accepts at least 2^{n-c} words. Yet the order of L_q is $o(2^n)$, by Corollary 19, which contradicts the fact that $(\psi_n)_{n\in\mathbb{N}}$ recognises L_q.

Now, suppose $\Sigma^* \setminus L_q \in \mathbf{SAC}^0$, again accepted by $(\psi_n)_{n\in\mathbb{N}}$ with constant c. Separate the positive from the negative literals in $\varphi_{1,n}$; there are at most $c' \leq c$ such positive literals. Thus, for any word $x = x_1 \cdots x_n \in \Sigma^*$, if $x_i = 1$ for all such positive literals, and $x_i = 0$ everywhere else, then ψ_n accepts x. But for large enough n, such x is in L_q by Proposition 11, which contradicts the fact that $(\psi_n)_{n\in\mathbb{N}}$ recognises $\Sigma^* \setminus L_q$. ◀

4.2 The Circuit Class $\bigoplus\mathbf{SAC}^0$

In this section, we consider the class $\bigoplus\mathbf{SAC}^0$, whose definition differs from that of $\mathbf{SAC}^0$ only in the choice of base set. Let $\oplus$ denote the XOR operation. As before, fix $q \in (0, 1/2)$.

▶ **Definition 40.** *A language $L \subset \{0,1\}^*$ is $\bigoplus\mathbf{SAC}^0$-recognisable if there exists a family $(C_i)_{i\in\mathbb{N}}$ of Boolean circuits which recognises L and which satisfies the following:*

1. *Each C_i is defined over the basic set $\{\wedge, \oplus\}$ and accepts negative literals.*
2. *The family $(C_i)_{i\in\mathbb{N}}$ has constant depth.*
3. *Each C_i has unbounded fan-in-$\oplus$ and bounded fan-in-$\wedge$.*
4. *Each C_i accepts words of length i.*

From this definition and the following observation, we can investigate languages larger than binary. Recall that in the previous subsection, we focussed solely on the two-element alphabet $\{0, 1\}$. This was forced by the fact that Boolean expressions have trouble expressing Boolean operations on non-binary languages (e.g. what does $0 \wedge 2$ evaluate to?). This can be remedied in the class $\bigoplus\mathbf{SAC}^0$ for some languages, courtesy of the operator $\oplus$.

It is readily seen that $(\{0, 1\}, \oplus, \wedge)$ is isomorphic to the field of two elements $\mathbb{F}_2 = (\mathbb{Z}/2\mathbb{Z}, +, \times)$. (Studying Boolean circuits in terms of the arithmetic of $\mathbb{F}_2$ goes back to Gál and Wigderson [8]. We also mention here similarities to the work of Razborov-Smolensky [31, 39, 38].) To extend this equivalence beyond binary alphabets, take the field $\mathbb{F}_p$ for some prime $p > 2$. By interpreting $(\oplus, \wedge)$ as $(+, \times)$ mod p, we extend $\mathbf{SAC}^0$-recognisability to alphabets of prime cardinality. Below, we give a natural characterisation of $\bigoplus\mathbf{SAC}^0$-recognisability in terms of propositional formulas, similar to that in Lemma 38.[8]

[8] For a classical definition of $\bigoplus\mathbf{SAC}^0$ in terms of the complexity of Boolean circuits see e.g. [20, Section 4].

▶ **Definition 41.** *Let $|\Sigma| = p$ for some $p \in \mathbb{P}$. Then L is $\bigoplus\mathbf{SAC}^0$-recognisable if there exists $c \in \mathbb{N}$ such that: for every $n \in \mathbb{N}$ and every $x \in \Sigma^n$ there exists $k_n \in \mathbb{N}$ and a formula $\psi_n = \bigoplus_{i=1}^{k_n} \varphi_{i,n}$ for which $\varphi_{i,n}$ is a conjunction of at most c literals and*

$$x \in L \iff \psi_n(x) \neq 0.$$

▶ Remark 42. Observe that there is a subtle difference between $\mathbf{SAC}^0$ and $\bigoplus\mathbf{SAC}^0$ in the case $p = 2$. An $\mathbf{SAC}^0$ circuit accepts a word $x \in \Sigma^n$ if *any* term in the disjunction of $\psi_n(x)$ holds. On the contrary, in $\bigoplus\mathbf{SAC}^0$, the disjunction is interpreted as addition modulo 2, and hence, x is accepted only if the number of terms in the disjunction of ψ_n is odd. Also, note that Definition 41 requires a real-world formalism in which gates are able to carry out addition and multiplication modulo p as a primitive. This assumption is not needed when $p = 2$, as such Boolean circuits can be modelled using $\oplus$ and $\wedge$, as mentioned.

For completeness, we mention here that $\mathbf{SAC}^0 \neq \mathbf{coSAC}^0$ (see [3]), while $\mathbf{co}\bigoplus\mathbf{SAC}^0 = \bigoplus\mathbf{SAC}^0$ (inverting a polynomial in a finite field requires only a constant number of layers; we use this fact in the proof of Theorem 46). Further, $\mathbf{SAC}^0 \not\subseteq \bigoplus\mathbf{SAC}^0$ [20, Theorem 39].

Below, we prove the following complexity characterisation of alphabets of prime cardinality.

▶ **Theorem 46.** *Let $|\Sigma| = p$ for some $p \in \mathbb{P}$. Then $L_q \notin \bigoplus\mathbf{SAC}^0$ and $\Sigma^* \setminus L_q \notin \bigoplus\mathbf{SAC}^0$.*

By translating prime-cardinality-alphabets into finite fields, we may use the tools of field theory. In this section, we collect facts about finite fields which we require to prove Theorem 46.

▶ **Lemma 43.** *Let $\mathbb{F}$ be a finite field.*

1. *By prime decomposition, $\mathbb{F}$ has prime characteristic.*
2. *$\mathbb{F}$ has order p^n for some $p \in \mathbb{P}$. [7, 33.2, 33.10]*
3. *If $\mathbb{F}$ has order p^n then $\mathbb{F}$ has characteristic p. [6, Sec. 14.3]*
4. *For every $p \in \mathbb{P}$ and $n \in \mathbb{N}$, there is one field up to isomorphism of order p^n [7, 33.12]. This field has a subfield of order p, the* prime subfield.
5. *All functions from $\mathbb{F}$ to itself are polynomial functions. [7, Exercises 22: 31.c.]*
6. *If $\mathbb{F}$ has order p^n and $x \in \mathbb{F}$ then $x^{p^m} = x^{p^{m+n}}$ for all $m \in \mathbb{N}$. In particular, $x = x^{p^n}$, since the multiplicative subgroup of $\mathbb{F}$ has order $p^n - 1$. [6, p. 550]*

If $p \in \mathbb{P}$ and $n \in \mathbb{N}$, let $\mathbb{F}_{p^n}$ denote the (unique up to isomorphism) field of order p^n.

▶ **Lemma 44.** *Suppose $\varphi\colon \mathbb{F}_{p^n} \to \mathbb{F}_p$ is linear, i.e. $\varphi(x+y) = \varphi(x)+\varphi(y)$ and $\varphi(ax) = a\varphi(x)$ for all $x, y \in \mathbb{F}_{p^n}$ and $a \in \mathbb{F}_p$. Then there exist $a_1, \ldots, a_n \in \mathbb{F}_{p^n}$ for which $\varphi(x) = \sum_{i=1}^{n} a_i x^{p^i}$. In fact, every linear function from $\mathbb{F}_{p^n}$ to $\mathbb{F}_p$ arises in this way.*

For a proof and related details on *field traces*, see for instance [24, Theorem 2.24] and [24, Chapter 2.3]. In fact, their proof shows that there exists *one* $z \in \mathbb{F}_{p^n}$ for which $a_i = z^{p^i}$. We now give a characterisation of $\bigoplus\mathbf{SAC}^0$ in terms of finite fields and their operations. This characterisation is akin to that of $\mathbf{SAC}^0$ in Lemma 38 in terms of propositional formulas.

▶ **Proposition 45.** *Let $\phi_n\colon \mathbb{F}_p^n \to \mathbb{F}_{p^n}$ be a linear isomorphism of vector spaces over $\mathbb{F}_p$, and suppose $L \subset \Sigma^*$ is $\bigoplus\mathbf{SAC}^0$-recognisable. Then there exists a family of polynomials $(f_n)_{n\in\mathbb{N}}$ with $f_n\colon \mathbb{F}_{p^n} \to \mathbb{F}_p$ for which*

$$x \in L \cap \Sigma^n \iff (f_n \circ \phi_n)(x) \neq 0$$

and for which there exists $\ell \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ we have $\deg(f_n) \leq p^n - p^{n-\ell}$.

For the proof, we direct the reader to [4]. We now combine the field-theoretic tools above to prove the main theorem of this section. Recall that $\mathbb{P}$ denotes the set of primes.

▶ **Theorem 46.** *Let $q \in (0, 1/2)$ and $|\Sigma| \in \mathbb{P}$. Then $L_q \notin \bigoplus\mathbf{SAC}^0$ and $\Sigma^* \setminus L_q \notin \bigoplus\mathbf{SAC}^0$.*

Proof. Suppose some circuit recognises L_q. By Proposition 45, there exists a family of polynomials (f_n) and $\ell \in \mathbb{N}$ for which $x \in L_q$ if and only if $(f_n \circ \phi_n)(x) \neq 0$ and $\deg(f_n) \leq p^n - p^{n-\ell}$. So, the number of roots of f_n – and, hence, the number of words not in L_q – is bounded above by $p^n - p^{n-\ell}$, so the cardinality of L_q is in $\Omega(p^n)$, contradicting Corollary 19.

For $\Sigma^* \setminus L_q$, note that the circuit can be augmented by a constant number of layers to flip the output of $f_n \circ \phi_n$ for any n (note that $\operatorname{ran}(f_n \circ \phi_n) \subseteq \mathbb{F}_p$). If $a_x = (f_n \circ \phi_n)(x) \neq 0$ then use Lemma 43 Item 6 to see that $a_x^p = a_x$; thus $a_x^{p-1} = 1$, and so the polynomial $\theta(x) = 1 - x^{p-1}$ satisfies $\theta(x) = 0 \iff a_x \neq 0$. As p is fixed, θ can be computed by a constant-depth circuit, which we may append to any $\bigoplus\mathbf{SAC}^0$-circuit recognising L_q to recognise $\Sigma^* \setminus L_q$. Since the former does not exist, neither does the latter. ◀

5 Open Questions

In this paper, we proved multiple results on the complexity of the measure A_{Ne} via the proxy family of sets $\{ L_q : q \in (0, 1/2) \}$. In particular, we showed that L_q is complicated from the viewpoint of pushdown automata (Theorems 34 and 35 and Corollary 36), and even certain Boolean circuits cannot recognise L_q, nor its complement (Theorems 39 and 46). We also proved the Shannon effect for A_{Ne} (Theorem 22). Pressing open questions pertain to refining these results on L_q – and, ultimately, to understanding the measure A_{Ne} even better.

In Section 4.2, and in Theorem 46 in particular, we considered alphabets of prime cardinality; however, our proof uses a non-standard definition of $\bigoplus\mathbf{SAC}^0$. We wonder:

▶ **Question 47.** Do the results from Theorem 46 apply to arbitrary alphabets using the definition of $\bigoplus\mathbf{SAC}^0$ given in Definition 40?

Finally, it is clear by definition that $A_{Ne}(x) \leq A_N(x)$ for all $x \in \Sigma^*$, for any finite alphabet Σ. This allows the extension of certain results from A_{Ne} to A_N. For instance, we note here that Theorems 33, 35, and 39 also apply to A_N since $A_{Ne}(x) \leq A_N(x)$. However, whether the equality $A_{Ne}(x) = A_N(x)$ holds in general remains the cardinal open question to fully understand the impact of exactness in Definition 7 compared to Definition 4.

▶ **Question 48.** Let $\Sigma = \{0, 1\}$. Does there exist $x \in \Sigma^*$ for which $A_{Ne}(x) < A_N(x)$?

Broadly, it is our hope that a better understanding of A_{Ne}, via proxies such as the L_q sets or otherwise, will lead to a better understanding of A_N, and (non-deterministic) automatic complexity in general. Similar to the comparison between regular languages being characterised by both NFAs and DFAs, a proof of $A_N = A_{Ne}$ would simplify computing the non-deterministic automatic complexity while preserving its naturalness as a measure of complexity. Conversely, $A_N \neq A_{Ne}$ would show that non-deterministic automatic complexity is dependent on the actual computation path, which would also be of philosophical interest.

References

1 Scott Aaronson. Complexity Zoo, 2025. URL: `https://complexityzoo.net/Complexity_Zoo` [cited 18 Apr 2025].

2 Achilles A. Beros, Bjørn Kjos-Hanssen, and Daylan Kaui Yogi. Planar digraphs for automatic complexity. In *Theory and applications of models of computation*, volume 11436 of *Lecture Notes in Comput. Sci.*, pages 59–73. Springer, Cham, 2019. `doi:10.1007/978-3-030-14812-6_5`.

3 Allan Borodin, Stephen A. Cook, Patrick W. Dymond, Walter L. Ruzzo, and Martin Tompa. Two applications of inductive counting for complementation problems. *SIAM J. Comput.*, 18(3):559–578, 1989. doi:10.1137/0218038.

4 Joey Chen, Bjørn Kjos-Hanssen, Ivan Koswara, Linus Richter, and Frank Stephan. Languages of Words of Low Automatic Complexity Are Hard to Compute, 2025. arXiv:2510.07696.

5 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic randomness and complexity*. Theory and Applications of Computability. Springer, New York, 2010. doi:10.1007/978-0-387-68441-3.

6 David S. Dummit and Richard M. Foote. *Abstract algebra*. John Wiley & Sons, Inc., Hoboken, NJ, third edition, 2004.

7 John B Fraleigh. *A first course in abstract algebra*. Pearson Education, Philadelphia, PA, 7th edition, 2003.

8 Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996. doi:10.1002/(sici)1098-2418(199608/09)9:1/2<99::aid-rsa7>3.3.co;2-o.

9 Seymour Ginsburg. *The mathematical theory of context-free languages*. McGraw-Hill Book Co., New York-London-Sydney, 1966.

10 Sheila A. Greibach. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *J. ACM*, 12(1):42–52, January 1965. doi:10.1145/321250.321254.

11 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation*. Pearson, Upper Saddle River, NJ, 3 edition, June 2006.

12 Kayleigh Hyde. Nondeterminstic Finite State Complexity. Master's thesis, University of Hawai'i at Mānoa, 2013. URL: http://hdl.handle.net/10125/29507.

13 Kayleigh K. Hyde and Bjørn Kjos-Hanssen. Nondeterministic automatic complexity of overlap-free and almost square-free words. *Electron. J. Combin.*, 22(3):Paper 3.22, 18, 2015. doi:10.37236/4851.

14 Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, volume 21 of *Progress in Computer Science and Applied Logic*. Birkhäuser Boston, Inc., Boston, MA, 2001. doi:10.1007/978-1-4612-0171-7.

15 Bjørn Kjos-Hanssen. On the complexity of automatic complexity. *Theory Comput. Syst.*, 61(4):1427–1439, 2017. doi:10.1007/s00224-017-9795-4.

16 Bjørn Kjos-Hanssen. Automatic complexity of shift register sequences. *Discrete Mathematics*, 341(9):2409–2417, 2018. doi:10.1016/j.disc.2018.05.015.

17 Bjørn Kjos-Hanssen. Automatic complexity of Fibonacci and Tribonacci words. *Discrete Applied Mathematics*, 289:446–454, 2021. doi:10.1016/j.dam.2020.10.014.

18 Bjørn Kjos-Hanssen. An incompressibility theorem for automatic complexity. *Forum Math. Sigma*, 9:e62, 7, 2021. doi:10.1017/fms.2021.58.

19 Bjørn Kjos-Hanssen. *Automatic complexity—a computable measure of irregularity*, volume 12 of *De Gruyter Series in Logic and its Applications*. De Gruyter, Berlin, [2024] ©2024. doi:10.1515/9783110774870.

20 Bjørn Kjos-Hanssen. Maximal automatic complexity and context-free languages. In *Aspects of computation and automata theory with applications*, volume 42 of *Lect. Notes Ser. Inst. Math. Sci. Natl. Univ. Singap.*, pages 335–352. World Sci. Publ., Hackensack, NJ, [2024] ©2024. doi:10.1142/9789811278631_0013.

21 A. N. Kolmogorov. Three approaches to the definition of the concept "quantity of information". *Problemy Peredači Informacii*, 1(vyp. 1):3–11, 1965. URL: https://www.mathnet.ru/eng/ppi68.

22 L. A. Levin. Laws of Information Conservation (Nongrowth) and Aspects of the Foundation of Probability Theory. *Problems Inform. Transmission*, 10(3):206–210, 1974. URL: https://www.mathnet.ru/eng/ppi1039.

23 Leonid A. Levin. Some theorems on the algorithmic approach to probability theory and information theory: (1971 dissertation directed by a.n. kolmogorov). *Annals of Pure and*

Applied Logic, 162(3):224–235, 2010. Special Issue: Dedicated to Nikolai Alexandrovich Shanin on the occasion of his 90th birthday. doi:10.1016/j.apal.2010.09.007.

24 Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. With a foreword by P. M. Cohn. doi:10.1017/CBO9780511525926.

25 O. B. Lupanov. The synthesis of contact circuits. *Dokl. Akad. Nauk SSSR (N.S.)*, 119:23–26, 1958.

26 O. B. Lupanov. The schemes of functional elements with delays. *Problemy Kibernet.*, (23):43–81, 303, 1970.

27 R. C. Lyndon and M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.*, 9:289–298, 1962. URL: http://projecteuclid.org/euclid.mmj/1028998766.

28 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966. doi:10.1016/S0019-9958(66)80018-9.

29 Jaban Meher and M. Ram Murty. Ramanujan's proof of Bertrand's postulate. *Amer. Math. Monthly*, 120(7):650–653, 2013. doi:10.4169/amer.math.monthly.120.07.650.

30 Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Faster Algorithms for Ranking/Unranking Bordered and Unbordered Words. In Zsuzsanna Lipták, Edleno Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *String Processing and Information Retrieval*, pages 257–271, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-72200-4_20.

31 A. A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41(4):333–338, April 1987. doi:10.1007/BF01137685.

32 Maria José Serna. Asymptotical behaviour of some non-uniform measures. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 23(3):281–293, 1989. doi:10.1051/ita/1989230302811.

33 Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, USA, first edition, 2008. doi:10.1017/CBO9780511808876.

34 Jeffrey Shallit and Yuri Breitbart. Automaticity I: Properties of a Measure of Descriptional Complexity. *Journal of Computer and System Sciences*, 53(1):10–25, 1996. doi:10.1006/jcss.1996.0046.

35 Jeffrey Shallit and Ming-Wei Wang. Automatic complexity of strings. *J. Autom. Lang. Comb.*, 6(4):537–554, 2001. 2nd Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (London, ON, 2000). doi:10.25596/jalc-2001-537.

36 Claude. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949. doi:10.1002/j.1538-7305.1949.tb03624.x.

37 Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 330–335, New York, NY, USA, 1983. Association for Computing Machinery. doi:10.1145/800061.808762.

38 R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 77–82, New York, NY, USA, 1987. Association for Computing Machinery. doi:10.1145/28395.28404.

39 R. Smolensky. On representations by low-degree polynomials. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 130–138, 1993. doi:10.1109/SFCS.1993.366874.

40 Ray J Solomonoff. A preliminary report on a general theory of inductive inference, February 1960. URL: https://raysolomonoff.com/publications/z138.pdf.

41 Ray J Solomonoff. A formal theory of inductive inference. Part I. *Information and control*, 7(1):1–22, 1964. doi:10.1016/S0019-9958(64)90223-2.

42 Ray J Solomonoff. A formal theory of inductive inference. Part II. *Information and control*, 7(2):224–254, 1964. doi:10.1016/S0019-9958(64)90131-7.

43 I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comput. Mach.*, 25(3):405–414, 1978. doi:10.1145/322077.322083.

44 H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. System Sci.*, 43(2):380–404, 1991. doi:10.1016/0022-0000(91)90020-6.

45 I. Wegener. *The Complexity of Boolean Functions*. Wiley Teubner on Applicable Theory in Computer Science. Wiley, 1987.

On the Interplay of Cube Learning and Dependency Schemes in QCDCL Proof Systems

Abhimanyu Choudhury
The Institute of Mathematical Sciences, Chennai, India
Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Meena Mahajan
The Institute of Mathematical Sciences, Chennai, India
Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai, India

Abstract

Quantified Conflict Driven Clause Leaning (QCDCL) is one of the main approaches to solving Quantified Boolean Formulas (QBF). Cube-learning is employed in this approach to ensure that true formulas can be verified. Dependency Schemes help to detect spurious dependencies that are implied by the variable ordering in the quantifier prefix of QBFs but are not essential for constructing (counter)models. This detection can provably shorten refutations in specific proof systems, and is expected to speed up runs of QBF solvers.

The simplest underlying proof system [BeyersdorffBöhm-LMCS2023], formalises the reasoning in the QCDCL approach on false formulas, when neither cube-learning nor dependency schemes is used. The work of [BöhmPeitlBeyersdorff-AI2024] further incorporates cube-learning. The work of [ChoudhuryMahajan-JAR2024] incorporates a limited use of dependency schemes, but without cube-learning.

In this work, proof systems underlying the reasoning of QCDCL solvers which use cube learning, and which use dependency schemes at all stages, are formalised. Sufficient conditions for soundness and completeness are presented, and it is shown that using the standard and reflexive resolution path dependency schemes ($\mathtt{D}^{\mathtt{std}}$ and $\mathtt{D}^{\mathtt{rrs}}$) to relax the decision order provably shortens refutations.

When the decisions are restricted to follow quantification order, but dependency schemes are used in propagation and learning, in conjunction with cube-learning, the resulting proof systems using the dependency schemes $\mathtt{D}^{\mathtt{std}}$ and $\mathtt{D}^{\mathtt{rrs}}$ are investigated in detail and their relative strengths are analysed.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases QBF, CDCL, Resolution, Dependency schemes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.25

Related Version *Full Version*: `https://arxiv.org/abs/2510.05876` [17]

Funding *Meena Mahajan*: Supported in part by the J. C. Bose Fellowship of SERB, ANRF.

1 Introduction

Despite the NP-hardness of the satisfiability problem, in the last three decades SAT solvers have been phenomenally successful in solving instances of humongous size, and have become the go-to tool in many practical industrial applications (see e.g. [30, 22]). This success has spurred ambitious programs to develop solvers for computationally even more hard problems. In particular, the PSPACE-complete problem of determining the truth of Quantified Boolean Formulas QBFs has many more applications (see e.g. [27]), and over the last twenty years QBF solvers have rapidly approached the state of industrial applicability.


© Abhimanyu Choudhury and Meena Mahajan;
licensed under Creative Commons License CC-BY 4.0



The paradigm that revolutionized SAT solving is Conflict Driven Clause Learning CDCL ([28]), and this is also one of the principal approaches (but not the only one) in QBF solving. The CDCL technique was lifted to QBFs in the form of QCDCL ([31], see also [18]; in [19], the term QDPLL is used), and implemented in state-of-the-art solvers DepQBF [20, 21] and Qute [23] with further augmentations to enhance performance.

For both SAT and QBF, solving techniques are intricately connected with proof systems. The runtime trace of a solver on a formula can be thought of as a proof of the final outcome (sat/unsat, true/false). Proof systems abstract out the reasoning employed in the solvers, and allow representing these traces-as-proofs as formal proofs. The CDCL paradigm in SAT solvers corresponds to resolution, a very well-studied proof system. There are multiple ways in which resolution can be lifted to QBFs, see [5] for an overview. As shown in [4], resolution proofs can be efficiently extracted from traces of CDCL-based SAT solvers. For QBFs, QCDCL traces yield proofs in the proof system long-distance Q-resolution $\mathtt{LDQ\text{-}Res}$ [31, 3]. However, the converse direction, going from resolution proofs to CDCL runs, famously shown for SAT in [25, 1], seemingly breaks down for QBF and QCDCL as currently implemented; the reasoning employed in basic QCDCL solvers was abstracted in [7, 15] as the proof systems $\mathtt{QCDCL}$ and $\mathtt{QCDCL}^{\mathtt{cube}}$, and shown to be exponentially weaker than $\mathtt{LDQ\text{-}Res}$.

The proof system $\mathtt{QCDCL}$ is a refutational proof system; it was formulated in [7] to explain the reasoning of basic QCDCL-style algorithms on false QBFs. The proof system $\mathtt{QCDCL}^{\mathtt{cube}}$, defined in [15], incorporates cube-learning as well, and can thus certify both falsity and truth of QBFs. Intriguingly, it was shown in [15] that even for false QBFs, where cube-learning is not necessary for completeness, it can still significantly shorten refutations. Very recently, it was shown in [8] that even when short refutations are actually found, it may take an exponentially long time to find them. Many other variants (different policies for decision order, propagations, reductions) have been studied extensively in [14].

One heuristic that has been used in some QCDCL solvers is the use of dependency schemes. These schemes involve performing some basic analysis of the formula structure and identifying spurious dependencies amongst variables, dependencies that are implied by the quantification order of variables but are not necessary for constructing (counter)models; see for instance [29]. Eliminating such dependencies would transform a QBF to a Dependency QBF, DQBF, for which the computational problem of deciding truth/falsity is even harder; it is NEXP-complete ([2, 12]). However, retaining the formulas as a QBF, and using information about spurious dependencies while propagating and learning, is still a feasible approach, that has been implemented in the solver DepQBF [20, 21] using the *standard dependency scheme* $\mathtt{D}^{\mathtt{std}}$. In resolution-based QBF proof systems, employing reduction rules based on the *reflexive resolution path dependency-scheme* $\mathtt{D}^{\mathtt{rrs}}$, is known to exponentially shorten refutations ([10]), and the expectation is that a similar advantage also manifests in QCDCL solvers.

This work makes progress towards formally understanding the strengths/limitations of using the dependency-scheme heuristic. The first steps in this direction were initiated in a recent work in [16]. It considered the simplest setting, in which the $\mathtt{QCDCL}$ proof system uses the $\mathtt{LEV\text{-}ORD}$ decision policy (deciding variables according to the quantification order), and does not learn cubes. A dependency scheme $\mathtt{D}$ is used in propagation by, and learning of, clauses. Additionally, a dependency scheme $\mathtt{D}'$ may be used to preprocess the formula, reducing all clauses according to $\mathtt{D}'$ before beginning the QCDCL trails. In this setting, when $\mathtt{D}$ and $\mathtt{D}'$ are "normal" schemes (as defined in [24]), the resulting proof systems were shown to be sound and refutationally complete. In the same setting, the four systems arising from using $\mathtt{D}^{\mathtt{rrs}}$ in preprocessing, in propagation/learning, in both, and in neither, were shown to

be incomparable with each other. In the underlying proof system LDQ-Res, using dependency information can never lengthen proofs. The handicap in QCDCL arises because QCDCL algorithms also need to search for the proof.

In this work, we consider more general settings. Our contributions are as follows:

Formalising intensive use of dependency schemes in QCDCL. We formalise the definitions of QCDCL and QCDCL$^{\text{cube}}$ proof systems that use dependency schemes more intensively: in the decision policy, which determines which variables can be "decided" at a particular stage, as well as in propagation and learning, with and without cube learning. Using a dependency scheme $\mathtt{D}_1$ in the decision policy means that a variable can be decided if all variables on which it depends, according to $\mathtt{D}_1$, are already assigned; this is the policy $\mathtt{D}_1$-ORD. Using a dependency scheme $\mathtt{D}_2$ in propagation and learning means that reductions enabled by $\mathtt{D}_2$ are performed whenever possible. For two dependency schemes $\mathtt{D}_1$ and $\mathtt{D}_2$ (which may be the same) we consider QCDCL$^{\text{cube}}$ proof systems that use $\mathtt{D}_1$ in the decision policy and $\mathtt{D}_2$ in propagation through and learning of clauses. We consider three scenarios with respect to cube-learning: (1) cube-learning is switched off completely; (2) cube propagation and learning is done without using any dependency schemes; or (3) cube propagation and learning use the scheme $\mathtt{D}_2$ but disallow long-distance term-resolution. The reason for this difference between clause and cube learning is that long-distance term resolution is not (yet?) known to be sound if used in conjunction with dependency schemes. We show that for normal $\mathtt{D}_1$, $\mathtt{D}_2$, the resulting systems are sound and refutationally complete; Theorem 3.12.

Provable advantage of D-ORD. We show that other parameters remaining the same, using either $\mathtt{D}^{\mathtt{rrs}}$ or $\mathtt{D}^{\mathtt{std}}$ as $\mathtt{D}_1$ is strictly better than using LEV-ORD; Theorem 3.13.

Using cube-learning, and dependency schemes only in propagation/learning. When the decision policy is restricted to LEV-ORD, we generalise the results from [16] to settings with cube-learning switched on, and also to settings where $\mathtt{D}^{\mathtt{std}}$ rather than $\mathtt{D}^{\mathtt{rrs}}$ is used. Specifically, we show that

1. Using $\mathtt{D}^{\mathtt{std}}$ in pre-processing is useless; Proposition 4.2.
2. Switching on cube learning provably adds power even in the presence of $\mathtt{D}^{\mathtt{std}}$ or $\mathtt{D}^{\mathtt{rrs}}$; Theorem 4.12.
3. Adding $\mathtt{D}^{\mathtt{rrs}}$ in various non-trivial ways to QCDCL$^{\text{cube}}$ results in proof systems that are not only pairwise incomparable, Theorem 4.14, but are also incomparable with both QCDCL$^{\text{cube}}$ and QCDCL; Theorems 4.13 and 4.15.
4. Adding $\mathtt{D}^{\mathtt{std}}$ to QCDCL is orthogonal to switching on cube learning; Theorem 4.16. In certain cases, adding $\mathtt{D}^{\mathtt{std}}$ to both QCDCL and QCDCL$^{\text{cube}}$ offers a provable advantage.
5. Although $\mathtt{D}^{\mathtt{rrs}}$ strictly refines $\mathtt{D}^{\mathtt{std}}$, in the context of QCDCL and QCDCL$^{\text{cube}}$, adding these schemes gives rise to incomparable systems; Theorem 4.17. Thus, the LEV-ORD policy can negate potential benefits of the strict refinement.

We use several known bounds on formulas from earlier works, and also show some new bounds for them. To obtain desired separations, we also introduce three carefully handcrafted new formulas. For easy reference, the known and new results (about previously defined and new formulas) are collated in Table 1 in Section 4. The known simulation order of the proof systems, incorporating prior known results as well as the new results proved here, are summarised in Figure 1, also in Section 4.

Organisation of the paper

In Section 2, we give some basic definitions and describe, in a high-level informal way, the background about known proof systems and dependency schemes. In Section 3, we define the new proof systems, show soundness and completeness, and show that the decision policy `D-ORD` is strictly more powerful than `LEV-ORD`. In Section 4, we briefly discuss preprocessing, we define three new QBF families and show various lower and upper bounds for their proof sizes, and we describe the simulation order among various `QCDCL` systems. We conclude with some pointers for further directions of interest. Detailed definitions from prior work, as well as detailed proofs of many new results, appear in the full version of this paper [17].

2 Preliminaries

Basic Notation

We assume familiarity with basic concepts from logic and computational complexity: Boolean variables and literals, clauses and cubes/terms, conjunctive normal form (CNF), substitutions, etc. We follow notation from [7, 15]; see also [5]. Selected relevant items are included here.

A literal ℓ is a Boolean variable x or its negation $\bar{x}$, and $\mathtt{var}(\ell)$ denotes the associated variable x. For a set S of clauses and a literal ℓ, we use shorthand $\ell \vee S$ to denote the set of clauses $\{\ell \vee C \mid C \in S\}$. The empty clause, denoted $\Box$, is unsatisfiable; the empty cube, denoted $\top$, is always true. A clause (cube) is said to be tautological (contradictory) if for some variable x it contains both x and $\bar{x}$. The resolution rule can be applied to clauses and to cubes. The resolvent of clauses $A' = A \vee x$ and $B' = B \vee \bar{x}$ is the clause $A \vee B$ denoted as $\mathtt{res}(A', B', x)$ or $\mathtt{res}(B', A', x)$. The resolvent of cubes $A' = A \wedge x$ and $B' = B \wedge \bar{x}$ is the cube $A \wedge B$, also denoted as $\mathtt{res}(A', B', x)$ or $\mathtt{res}(B', A', x)$.

A Quantified Boolean Formula (QBF) in prenex conjunction normal form (PCNF) is a prefix with a list of variables, each quantified either existentially or universally, and a matrix, which is a set (conjunction) of clauses over these variables. That is, it has the form $\Phi = \mathcal{Q}\vec{x} \cdot \varphi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \;\; \varphi(x_1, x_2, \dots, x_n)$ where φ is a propositional formula in CNF, and each Q_i is in $\{\exists, \forall\}$. We denote by $X_\exists$ ($X_\forall$ respectively) the set of all variables quantified existentially (resp. universally). A QBF is true if for each variable $x_i \in X_\exists$, there exists a (Skolem) function s_i, depending only on variables $x_j \in X_\forall$ with $j < i$, such that substituting these s_i in φ yields a tautology. Similarly, the formula is false if for each variable $x_i \in X_\forall$, there is a (Herbrand) function h_i, depending only on variables $x_j \in X_\exists$ with $j < i$, such that substituting h_i in φ yields an unsatisfiable formula.

Some QBF proof systems, and the Dependency Scheme heuristic

The propositional proof system Resolution certifies unsatisfiability of a propositional formula by adding clauses obtaining through resolution until the empty clause is added. This can be lifted to QBFs in many ways. One of the simplest ways is to use the resolution rule along with a universal reduction rule, that allows removing a universal literal u or $\bar{u}$ from a clause if it is not '*blocked*; that is, the clause has no existential literals quantified after u in the prefix. This gives rise to the system `QU-Res`; its restriction where resolution is allowed only on existential pivots is the system `Q-Res`. The long-distance resolution rule generalises resolution by permitting seemingly useless universal tautological clauses under certain side-conditions, and gives rise to the system `LDQ-Res` that generalises `Q-Res`. Informally, in this system, a resolution on x is permitted even if the resolvent ends up having u and $\bar{u}$ for some universal

variable u, provided u is quantified to the right of x. The presence of u and $\bar{u}$ together, often referred to as a *merged literal* u^*, is to be interpreted not as a tautology but as a place-holder for a partial strategy for u depending on the value of the pivot x.

In direct analogy to Q-Res and LDQ-Res are the proof systems Q-TermRes and LDQ-TermRes for certifying truth. Here the resolution is performed on terms, or cubes, with universal pivots, and existential literals can be reduced from a term if not blocked by universal literals quantified after them. The goal is to derive the empty term. A key point of difference is at the start; since the QBF is in PCNF, there are no terms to begin with. The Axiom rule in these systems permits starting with any term that satisfies the matrix.

For formal definitions of these proof systems, see for instance, Figure 2 in [9] (for Q-Res,QU-Res, LDQ-Res), Figure 2, in [29] (for Q-TermRes).

The notion of blocking, used in the reduction rules, stems from the understanding that if variables x, y are quantified differently with y quantified after x, then the value of y in a (counter)model may non-trivially *depend* on x. If there is no literal blocking x, then the satisfaction of the clause (falsification of cube) should not rely on the unblocked universal (resp. existential) x. The dependency scheme heuristic refines this further. If a syntactic examination of the clause-variable structure can reveal that y does not really depend on x, *even though it is quantified later*, then x can be reduced even in the presence of y. This can drive the process towards the empty clause/term faster. Dependency schemes do precisely this. They identify pairs (x, y) where x and y are quantified differently and where y can be safely assumed to be independent of x. (Actually, they list pairs where y may depend on x; the other pairs can be assumed to be independent.) The trivial dependency scheme associates with each QBF Φ the dependency set $\mathtt{D}^{\mathtt{trv}}(\Phi) = \{(x, y) \mid y \text{ is quantifed after, and differently from, } x\}$. Other schemes can associate subsets of this set. The schemes relevant to this paper are the standard scheme $\mathtt{D}^{\mathtt{std}}$ (see Def 7 in [26] and Def 9 in [29]), and the reflexive resolution scheme $\mathtt{D}^{\mathtt{rrs}}$ (see Defs 3,4,6 in [29]). For every Φ, $\mathtt{D}^{\mathtt{trv}}(\Phi) \supseteq \mathtt{D}^{\mathtt{std}}(\Phi) \supseteq \mathtt{D}^{\mathtt{rrs}}(\Phi)$.

When a dependency scheme D is incorporated into any of the preceding proof systems, the reduction rule becomes more generally applicable, and the side-conditions concerning merged literals also become more permissive. This gives rise to the proof systems Q(D)-Res, LDQ(D)-Res, Q(D)-TermRes, LDQ(D)-TermRes. See, for instance, Figure 3 and Section 3.3 in [29] (for Q-TermRes and D-reductions), and Figure 1 in [24] (for LDQ(D)-Res). For a clause and a dependency scheme D, we denote by red-D(C) the clause obtained by removing all unblocked universal literals from C. Similarly, for a cube C, red-D$_\exists(C)$ denotes the cube obtained by removing all unblocked existential literals from C.

An important subclass of dependency schemes are the so-called *normal* dependency schemes, which have the property of being "monotone" and "simple". See Def. 7 in [24] for the precise definition. The dependency schemes $\mathtt{D}^{\mathtt{trv}}$, $\mathtt{D}^{\mathtt{std}}$, $\mathtt{D}^{\mathtt{rrs}}$ are all normal. This class of schemes is of interest to us because it is known that for normal dependency schemes, Q(D)-Res and LDQ(D)-Res are sound and refutationally complete [24]. The system Q(D)-TermRes is known to be sound and complete on true formulas for $\mathtt{D} \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{rrs}}\}$ (in [29], soundness is shown for a stronger dependency scheme, $\mathtt{D}^{\mathtt{res}}$, implying soundness for $\mathtt{D}^{\mathtt{rrs}}$ as well). However the soundness of LDQ(D)-TermRes is not known for dependency schemes other than $\mathtt{D}^{\mathtt{trv}}$.

We say proof system $\mathtt{P}_1$ simulates a proof system $\mathtt{P}_2$ if some computable function transforms proofs in $\mathtt{P}_2$ into proofs in $\mathtt{P}_1$ with at most polynomial blow-up in proof size. If this function is also computable in polynomial time (in the given proof size), we say that $\mathtt{P}_1$ p-simulates $\mathtt{P}_2$. Two systems are said to be incomparable if neither simulates the other.

By definition LDQ(D)-Res p-simulates LDQ-Res and Q(D)-Res, both of which p-simulate Q-Res. Similarly, both LDQ-TermRes and Q(D)-TermRes p-simulate Q-TermRes. It is also known that Q(D)-Res is exponentially stronger than Q-Res for $\mathtt{D} = \mathtt{D}^{\mathtt{rrs}}$; [10, 11].

The proof systems QCDCL with and without cube learning

The proof system for QCDCL, as defined in [7], formalizes reasoning in QCDCL algorithms operating on false formulas without cube learning. These algorithms construct *trails* or partial assignments in a specific way – *decide* values of variables according to some policy, *propagate* values of existential variables that appear in clauses which become unit after restriction by the trail so far and by universal reduction (call such a clause the *antecedent* of the propagated literal) – trying to satisfy the matrix. If a conflict is reached, then the trail is inconsistent with any Skolem function. *Conflict analysis* is performed, and a new clause is *learnt* and added to the matrix. If the empty clause is learned, the formula is deemed false. The corresponding refutation in the proof system QCDCL consists of the sequence of constructed trails, and for each trail the sequence of long-distance resolution steps performed in conflict analysis to learn a clause. The full definition can be found in [7] (Def. 3.5).

The above formulation of the QCDCL system only considers trails that end in a conflict. Trails ending in a satisfying assignment are ignored. This is enough to ensure refutational completeness – the ability to prove all false QBFs false. However, from satisfying assignments, solvers can and do learn cubes (or terms), and this is necessary to prove true QBFs true. In [15] it was shown that even while refuting false QBFs, allowing cube learning from satisfying assignments can be advantageous. This led to the definition of the proof system $\mathtt{QCDCL}^{\mathtt{cube}}$, and in [15], it was shown to be strictly stronger than the standard QCDCL system. The main idea in cube learning systems is to consider satisfying assignments also as conflicts, albeit of a different kind, and to learn cubes from these conflicts. (The algorithm learns that such a trail is inconsistent with any Herbrand function.) Learnt cubes are added disjunctively to the matrix, which thus at intermediate stages is not necessarily in CNF but is the disjunction of a CNF formula and some cubes. With the augmented CNF matrices, cubes that become unit after existential reduction can now propagate universal variables in a way that falsifies the cube. Also, with the augmented CNF matrices, a trail may end up satisfying a cube rather than the CNF; this too is now a conflict, and conflict analysis involving term-resolution can be performed to learn a new cube. For formal details, see Section 3 in [15].

3 Adding dependency schemes to the QCDCL proof system

The work done in [16] was the first to formalise the addition of dependency schemes to the QCDCL proof system. It was done only for the setting where trails follow level-ordered decisions, and there is no cube learning. Here we relax both these restrictions; we allow dependency schemes to affect the decision policy of the trail, and we allow cube-learning.

Dependency schemes can be used in QCDCL algorithms in many ways:
(1) in specifying the decision order, (2) in specifying how reduction, propagation, and learning of clauses are performed, and (3) in specifying how these are performed for cubes, if at all. We set up unified notation to describe all such QCDCL-based proof systems.

▶ **Definition 3.1** (D-ORD). *For a dependency scheme* D*, the decision policy* D-ORD *permits a decision on a variable* x *at some point in a trail if all variables* y *on which* x *depends according to* D *(i.e.* $(y, x) \in \mathtt{D}$*), have already been assigned.*
The specific case of LEV-ORD *is the policy where decisions must respect quantification level order;* ANY-ORD *is the policy where decisions are unrestricted.*

▶ **Definition 3.2.** $\mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{ClausePol}, \mathtt{CubePol})$ *is the* QCDCL *proof system where*

1. ORD *denotes the decision policy; e.g.* LEV-ORD, D′-ORD, ANY-ORD.
   In D′-ORD, D′ *denotes the dependency scheme, such as* $\mathtt{D}^{\mathtt{rrs}}$-ORD, $\mathtt{D}^{\mathtt{std}}$-ORD.

2. `ClausePol` *is the dependency scheme* `D` *used in reduction, propagation, and learning for clauses. Note that this scheme need not be the same as the* $\mathtt{D}'$ *in* $\mathtt{D}'$`-ORD`.
3. `CubePol` $\in \{$`No-Cube`, `Cube-LD`, `Cube-D`$\}$ *denotes the type of usage of cube learning;*
   a. `No-Cube`*: No cube learning.*
   b. `Cube-LD`*: Cube Learning used, but no dependency scheme (only* $\mathtt{D}^{\mathtt{trv}}$*) in cube propagation, and cube learning using* `LDQ-TermRes`.
   c. `Cube-D`*: Cube Learning used, dependency scheme* `D` *used in propagation, and cube learning is done using* `Q(D)-TermRes`.

Note that in this notation, clause learning always uses `LDQ(D)-Res`, where `D` might well be $\mathtt{D}^{\mathtt{trv}}$. However, for cube learning, the propagation and learning can use either long-distance term resolution `LDQ(D)-TermRes`, or dependency schemes without long-distance `Q(D)-TermRes`, not both. We impose this condition because the soundness of `LDQ(D)-TermRes` is not known.

In the notation of Definition 3.2, the standard vanilla `QCDCL` system denoted $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}_{\mathtt{RED}}$ in [7] would be $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$, whereas the $\mathtt{QCDCL}^{\mathtt{cube}}$ system from [15] would be $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$. Further, the dependency-based system $\mathtt{QCDCL}(\mathtt{D}^{\mathtt{rrs}})$ introduced in [16] would be $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$.

To define what a derivation in these `QCDCL` systems looks like, we first define trails and the learnable clauses and cubes from trails in this system. The following definitions are the natural generalisations of the corresponding ones from [7, 15]. We give a short illustration in Example 3.6 after Definition 3.5.

The trail is a sequence of literals (or $\Box$, $\top$)

$$T = (p_{(0,1)}, \cdots, p_{(0,g_0)}; \mathbf{d_1}, p_{(1,1)}, \cdots p_{(1,g_1)}; \mathbf{d_2}, \cdots\cdots\cdots; \mathbf{d_r}, p_{(r,1)}, \cdots p_{(r,g_r)})$$

where the literals d_i (in boldface) are decision literals, the literals $p_{i,j}$ are propagated literals. and no opposing literals appear. We can also view it as a set of literals or an assignment, and the corresponding clause (cube) is the disjunction (conjunction) of all literals in it.

The learnable constraints from a trail are defined as follows:

▶ **Definition 3.3** (learning from conflict). *From a trail*

$$T = (p_{(0,1)}, \cdots, p_{(0,g_0)}; \mathbf{d_1}, p_{(1,1)}, \cdots p_{(1,g_1)}; \mathbf{d_2}, \cdots\cdots\cdots; \mathbf{d_r}, p_{(r,1)}, \cdots p_{(r,g_r)})$$

ending in a conflict $p_{(r,g_r)} \in \{\Box, \top\}$*, the set* L_T *of learnable constraints has a clause* $C_{(i,j)}$ *associated with each propagated literal* $p_{(i,j)}$ *propagated in the trail if* $p_{(r,g_r)} = \Box$*, and a cube associated with each propagated literal in the trail if* $p_{(r,g_r)} = \top$*. These associated clauses/cubes are constructed by tracing the conflict backwards through the trail as follows. (*$\mathtt{ante}(\ell)$ *denotes the clause/cube that causes literal* ℓ *to be propagated; i.e. the antecedent.) Starting with* $\mathtt{ante}(\Box)$ *(respectively* $\mathtt{ante}(\top)$*), we resolve in reverse over the antecedent clauses (cubes) that propagated the existential (universal) variables as described below. All such resulting clauses (cubes) are learnable constraints.*

In particular, if $p_{(r,g_r)} = \Box$*, then*

- $C_{(r,g_r)} = \mathtt{red\text{-}D}(\mathtt{ante}(p_{(r,g_r)}))$.
- *For* $i \in \{0, 1, \cdots, r\}$ *and* $j \in [g_i - 1]$*, if* $\mathtt{var}(p_{(i,j)}) \in X_\exists$ *and* $\bar{p}_{(i,j)} \in C_{(i,j+1)}$*, then* $C_{(i,j)}$ *is the clause obtained by resolving the clause* $C_{(i,j+1)}$ *with the clause obtained from the antecedent of* $p_{(i,j)}$ *after reduction; such a resolution is possible on pivot* $p_{(i,j)}$*. Otherwise,* $C_{(i,j)}$ *is simply the same as* $C_{(i,j+1)}$*. Thus,* $C_{(i,j)}$ *equals* $\mathtt{red\text{-}D}[\mathtt{res}(C_{(i,j+1)}, \mathtt{red\text{-}D}(\mathtt{ante}(p_{(i,j)})), p_{(i,j)})]$ *if* $\mathtt{var}(p_{(i,j)}) \in X_\exists$ *and* $\bar{p}_{(i,j)} \in C_{(i,j+1)}$*, and is* $C_{(i,j+1)}$ *otherwise.*

- *The learning process skips decision variables, so* $C_{(i,g_i)}$ *is defined using* $p_{(i,g_i)}$ *and* $C_{(i+1,1)}$. *For* $i \in \{0, 1, \cdots, r-1\}$. $C_{(i,g_i)}$ *equals* $\texttt{red-D}[\texttt{res}(C_{(i+1,1)}, \texttt{red-D}(\texttt{ante}(p_{(i,g_i)})), p_{(i,g_i)})]$ *if* $\texttt{var}(p_{(i,g_i)}) \in X_\exists$ *and* $\bar{p}_{(i,g_i)} \in C_{(i+1,1)}$, *and is* $C_{(i+1,1)}$ *otherwise.*

If $p_{(r,g_r)} = \top$, *then non-trivial cube-resolution is performed when* $p_{(i,j)}$ *is universal, not existential. The set* L_T *depends on* CubePol. *If* CubePol = Cube-LD, *then* red-D *is replaced by* $\texttt{red}_\exists$. *If* CubePol = Cube-D, *then* red-D *is replaced by* $\texttt{red-D}_\exists$, *but the* res *step is performed only if it is a valid* Q(D)-TermRes *resolution step; otherwise we use the previously learnt cube, just as we do in clause learning when the resolution on* $p_{(i,j)}$ *is not defined.*

▶ **Definition 3.4** (learning from satisfaction). *From a trail* T *that assigns all variables, satisfies all clauses, and does not satisfy any cube, the set of learnable constraints is defined as follows: For any set* L *of literals, let* t_L *denote the cube that is the conjunction of all literals in* L. *Viewing the trail* T *as a set of literals,*

$$L_T = \begin{cases} \{\texttt{red}_\exists(t_{T'}) \mid T' \subseteq T;\ T' \text{ satisfies all clauses}\} & \text{if } \texttt{CubePol} = \texttt{Cube-LD} \\ \{\texttt{red-D}_\exists(t_{T'}) \mid T' \subseteq T;\ T' \text{ satisfies all clauses}\} & \text{if } \texttt{CubePol} = \texttt{Cube-D}. \end{cases}$$

▶ **Definition 3.5.** *For a specific choice of* ORD, ClausePol, CubePol, *let* P *be the proof system* $\texttt{QCDCL}^{\texttt{ORD}}(\texttt{ClausePol}, \texttt{CubePol})$. *A* P*-derivation* ι *from a PCNF QBF* $\Phi = \mathcal{Q}\vec{x} \cdot \varphi$ *of a clause or cube* C *is a sequence* ι *of triples,* $\iota = (T_1, C_1, \pi_1), \cdots, (T_m, C_m, \pi_m)$, *where each* T_i *is a trail, each* C_i *is a clause/cube, and* $C_m = C$. *The objects* T_i, C_i, π_i *are as defined below.*

For each $i \in [m]$, φ_i *is a propositional formula of the form* $\varphi_i = \left(\bigwedge_{C \in \mathcal{C}_i} C\right) \vee \left(\bigvee_{T \in \mathcal{T}_i} T\right)$, *where* $\mathcal{C}_i$ *is a set of clauses and* $\mathcal{T}_i$ *is a set of cubes. These formulas are defined iteratively; initially we have all the clauses of* φ *and no terms, and after each trail either a clause or a term is learnt and added. Formally,*

$$\begin{array}{rrlrl} & \mathcal{C}_1 & = \{C \mid C \in \varphi\}, & \mathcal{T}_1 & = \emptyset \\ \text{If } C_i \text{ is a clause:} & \mathcal{C}_{i+1} & = \mathcal{C}_i \cup \{C_i\}, & \mathcal{T}_{i+1} & = \mathcal{T}_i. \\ \text{If } C_i \text{ is a cube:} & \mathcal{C}_{i+1} & = \mathcal{C}_i, & \mathcal{T}_{i+1} & = \mathcal{T}_i \cup \{C_i\}. \end{array}$$

For each $i \in [m]$, Φ_i *is the QBF with the same quantifier prefix as* Φ, *and inner formula* φ_i. *For each* $i \in [m]$, T_i *is a trail from the formula* Φ_i, C_i *is a learnable clause or cube from* T_i, *and* π_i *is the derivation of* C_i *from* Φ *in the system as per* ClausePol *or* CubePol.

A refutation in these systems is a derivation of the empty clause $\square$, *and a verification in this system is a derivation of the empty term* $\top$.

▶ **Example 3.6.** The $\texttt{TwoPHPandCT}_n$ formulas, defined in [16](Section 4.5), have the prefix $\mathcal{Q} = \forall u \exists x_1 \cdots x_{s_n}\ \exists y_1 \cdots y_{s_n}\ \forall v \exists z_1, z_2$ and the matrix

$$\begin{array}{c} u \vee \texttt{PHP}_n(x_1, \cdots, x_{s_n}) \qquad \bar{u} \vee \texttt{PHP}(y_1, \cdots, y_{s_n}) \\ v \vee z_1 \vee z_2\ ,\ v \vee \bar{z}_1 \vee z_2\ ,\ v \vee z_1 \vee \bar{z}_2\ ,\ v \vee \bar{z}_1 \vee \bar{z}_2 \end{array}$$

Here $\texttt{PHP}_n$ refers to the propositional Pigeon-hole-principle formulas that assert the existence of a map from $n+1$ pigeons to n holes without collision; these formulas are known to be exponentially hard for resolution. Due to $\texttt{PHP}_n$, the matrix is unsatisfiable, and thus cube-learning makes no difference. So we consider No-Cube.

Consider refutations using D-ORD, with $\texttt{D} = \texttt{D}^{\texttt{rrs}}$ or $\texttt{D}^{\texttt{std}}$. For these formulas, it can be seen that $\texttt{D}^{\texttt{rrs}} = \emptyset$ and $\texttt{D}^{\texttt{std}} = \{(u, x_i), (u, y_i) :\ \text{for all } i\} \cup \{(v, z_1), (v, z_2)\}$. The v, z_1, z_2 variables are completely independent from the u, x, y variables; neither (x_i, v) nor (y_i, v) is in D for any i. Therefore using the D-ORD decision policy, we can decide v or z_i in the beginning. Consider the trail that decides to assign v to false and then z_1 to true. The clause $v \vee \bar{z}_1 \vee z_2$ becomes unit, so z_2 is propagated. Then the clause $v \vee \bar{z}_1 \vee \bar{z}_2$ becomes empty, and the empty clause is

propagated, leading to conflict. That is, in $T_1 = \bar{\mathbf{v}}; \mathbf{z_1}, z_2, \square$, we have $\texttt{ante}(\square) = v \vee \bar{z}_1 \vee \bar{z}_2$, and $\texttt{ante}(z_2) = v \vee \bar{z}_1 \vee z_2$. In conflict analysis, we first resolve (the reduced version of) $\texttt{ante}(\square)$ with $\texttt{ante}(z_2)$ to obtain $v \vee \bar{z}_1$.

If $\texttt{D}^{\texttt{rrs}}$ is used in learning, then this can be reduced further to $\bar{z}_1$, which is learnt, i.e. included in $\mathcal{C}_2$ and Φ_2. The next trail then begins with the propagaed literal $\bar{z}_1$, and propagates further; $T_2 = \bar{z}_1, z_2, \square$, and $\texttt{ante}(\square) = v \vee z_1 \vee \bar{z}_2$, $\texttt{ante}(z_2) = v \vee z_1 \vee z_2$, $\texttt{ante}(\bar{z}_1) = \bar{z}_1$, allowing us to learn $\square$. This is a refutation in $\texttt{QCDCL}^{\texttt{D}^{\texttt{rrs}}}(\texttt{D}^{\texttt{rrs}}, \texttt{No-Cube})$ or $\texttt{QCDCL}^{\texttt{D}^{\texttt{std}}}(\texttt{D}^{\texttt{rrs}}, \texttt{No-Cube})$, but not in $\texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{rrs}}, \texttt{No-Cube})$.

If $\texttt{D}^{\texttt{std}}$ or $\texttt{D}^{\texttt{trv}}$ is used in learning, then from trail T_1 the clause $v \vee \bar{z}_1$ is learnt (i.e. included in $\mathcal{C}_2$ and Φ_2) since it cannot be further reduced. The next trail must again begin with a decision. With $T_2 = \bar{\mathbf{v}}, \bar{z}_1, z_2, \square$, $\texttt{ante}(\square) = v \vee z_1 \vee \bar{z}_2$, $\texttt{ante}(z_2) = v \vee z_1 \vee z_2$, $\texttt{ante}(\bar{z}_1) = v \vee \bar{z}_1$, allowing us to learn $\square$. This is a refutation in $\texttt{QCDCL}^{\texttt{D}_1}(\texttt{D}_2, \texttt{No-Cube})$, where $\texttt{D}_1$ could be $\texttt{D}^{\texttt{rrs}}$ or $\texttt{D}^{\texttt{std}}$ but not $\texttt{D}^{\texttt{trv}}$, and $\texttt{D}_2$ could be $\texttt{D}^{\texttt{std}}$ or $\texttt{D}^{\texttt{trv}}$.

By definition, D-ORD generalises LEV-ORD, and ANY-ORD generalises D-ORD. Thus,

▶ **Observation 3.7.** *For a* $\texttt{CubePol} \in \{\texttt{No-Cube}, \texttt{Cube-LD}, \texttt{Cube-D}\}$*, and dependency schemes* $\texttt{D}, \texttt{D}'$*,*

- *Every derivation in* $\texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}, \texttt{CubePol})$ *is a derivation in* $\texttt{QCDCL}^{\texttt{D}'\texttt{-ORD}}(\texttt{D}, \texttt{CubePol})$*.*
- *Every derivation in* $\texttt{QCDCL}^{\texttt{D}'\texttt{-ORD}}(\texttt{D}, \texttt{CubePol})$ *is a derivation in* $\texttt{QCDCL}^{\texttt{ANY-ORD}}(\texttt{D}, \texttt{CubePol})$*.*

Trails in a system without cube learning are also trails in the corresponding system with cube learning. Hence:

▶ **Observation 3.8.** *For* $\texttt{CubePol} \in \{\texttt{Cube-LD}, \texttt{Cube-D}\}$*, and for any decision policy* ORD*, any derivation in* $\texttt{QCDCL}^{\texttt{ORD}}(\texttt{D}, \texttt{No-Cube})$ *is also a derivation in* $\texttt{QCDCL}^{\texttt{ORD}}(\texttt{D}, \texttt{CubePol})$*.*

If the matrix of the given PCNF formula is *unsatisfiable*, then no satisfying trail can ever be constructed, no matter what policy is used, so no "cube learning" can happen. Hence:

▶ **Observation 3.9.** *For* $\texttt{CubePol} \in \{\texttt{Cube-LD}, \texttt{Cube-D}\}$*, and for any decision policy* ORD*, if a PCNF formula* Φ *has an unsatisfiable matrix, then any derivation from* Φ *in the system* $\texttt{QCDCL}^{\texttt{ORD}}(\texttt{D}, \texttt{CubePol})$ *is also a derivation in* $\texttt{QCDCL}^{\texttt{ORD}}(\texttt{D}, \texttt{No-Cube})$*.*

The following result is shown in [16].

▶ **Proposition 3.10** (Theorem 1 in [16])**.** *For any normal dependency scheme* D*, the system* $\texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}, \texttt{No-Cube})$ *is refutationally complete.*

A minor adaptation of the proof of Theorem 3.9 in [7] shows the following soundness:

▶ **Theorem 3.11.** *For any normal dependency scheme* D*, the proof systems* $\texttt{QCDCL}^{\texttt{ANY-ORD}}(\texttt{D}, \texttt{Cube-LD})$ *and* $\texttt{QCDCL}^{\texttt{ANY-ORD}}(\texttt{D}, \texttt{Cube-D})$ *are sound.*

Proof. To show that both these systems are sound, it is enough to show the following three statements: (1) The derivation of any learnt clause is a valid LDQ(D)-Res derivation. (2) If $\texttt{CubePol} = \texttt{Cube-LD}$, the derivation of any learnt cube is a valid LDQ-TermRes derivation, and the addition of cubes when learning from satisfaction is sound. (3) If $\texttt{CubePol} = \texttt{Cube-D}$, the derivation of any learnt cube is a valid Q(D)-TermRes derivation. From these three it follows that sticking together the derivations of the final learnt empty clause/term gives a proof in the corresponding system, and all these systems are known to be sound.

Statement (3) is true by definition: term resolution in learning is performed only if it is valid in Q(D)-TermRes. For Statement (2), cube learning is shown to be sound in [15, Theorem 3.8]. For statement (1), we need to show that the resolution steps performed while

learning respect the side-conditions of $\mathtt{LDRes}(\mathtt{D})$. The analogous statement when $\mathtt{D} = \mathtt{D}^{\mathtt{trv}}$ is proved in [7, Lemma 3.7, Proposition 3.8, Theorem 3.9], but the same proof works with any $\mathtt{D}$. For details, see the full version [17]. ◀

From Proposition 3.10, Theorem 3.11, Observation 3.7 and Observation 3.8, we conclude that all the aforementioned systems are sound and refutationally complete.

▶ **Theorem 3.12.** *For any dependency schemes* $\mathtt{D}'$, $\mathtt{D}$ *where* $\mathtt{D}$ *is normal, and for each* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}\}$, *the proof system* $\mathtt{QCDCL}^{\mathtt{D}'\text{-}\mathtt{ORD}}(\mathtt{D}, \mathtt{CubePol})$ *is sound and refutationally complete.*

In QCDCL based proof systems, incorporating dependency schemes into propagation and learning processes does not always yield benefits: as shown in [16], certain pathological formulas can render the addition of dependency schemes disadvantageous when decisions are constrained to the LEV-ORD decision policy.

If the dependency shceme is allowed to influence the decision order (i.e., the system adopts the D-ORD decision policy), we show below that the resulting systems are strictly more powerful than their counterparts using LEV-ORD. (For $\mathtt{D}_1 = \mathtt{D}^{\mathtt{std}}$, an advantage over LEV-ORD was noted already in [19].) However, they remain strictly weaker than the $\mathtt{LDQ}(\mathtt{D})\text{-}\mathtt{Res}$ systems. (For the proofs, see the full version [17].)

▶ **Theorem 3.13.** *For dependency schemes* $\mathtt{D}_1 \in \{\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{std}}\}$ *and* $\mathtt{D}_2 \in \{\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{std}}, \mathtt{D}^{\mathtt{trv}}\}$, *and for policy* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}_2\}$, *the proof system* $\mathtt{QCDCL}^{\mathtt{D}_1\text{-}\mathtt{ORD}}(\mathtt{D}_2, \mathtt{CubePol})$ *p-simulates* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}_2, \mathtt{CubePol})$ *and is not simulated by it.*

▶ **Theorem 3.14.** *For* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}\}$ *in* $\mathtt{D} \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{std}}, \mathtt{D}^{\mathtt{rrs}}\}$, *the proof system* $\mathtt{LDQ}(\mathtt{D})\text{-}\mathtt{Res}$ *p-simulates* $\mathtt{QCDCL}^{\mathtt{D}\text{-}\mathtt{ORD}}(\mathtt{D}, \mathtt{CubePol})$ *and is not simulated by it.*

4 Dependency-schemes-based QCDCL systems restricted to LEV-ORD

We now restrict our attention to proof systems utilizing only the LEV-ORD decision policy. Even with LEV-ORD, dependency schemes can affect (enhance or impair) the performance of QCDCL systems. These are the systems also considered in [7, 15, 16].

In [16], another way of incorporating dependency schemes was also considered, through "preprocessing". For a more complete comparison, we very briefly define such systems here as well. Preprocessing a formula using a dependency scheme simply means applying all reductions enabled by it on the input formula, and then proceeding with whatever version of QCDCL is of interest, on the reduced formula.

▶ **Definition 4.1.** *For a QBF* $\Phi = \mathcal{Q} \cdot \phi$ *and a normal dependency scheme* $\mathtt{D}$, *a derivation of a clause* C *from* Φ *in* $\mathtt{D} + \mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{ClausePol}, \mathtt{CubePol})$ *is a derivation of* C *from the QBF* $\Psi = \mathtt{red\text{-}D}(\Phi)$ *in* $\mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{ClausePol}, \mathtt{CubePol})$.

As shown in Theorem 4 of [16], preprocessing using $\mathtt{D}^{\mathtt{rrs}}$ can significantly alter the system strength. In contrast, we observe below that preprocessing using schemes $\mathtt{D}^{\mathtt{trv}}$ or $\mathtt{D}^{\mathtt{std}}$ has no effect, no matter what version of QCDCL is the subsequent system.

▶ **Proposition 4.2.** *For every decision policy* ORD, *dependency scheme* $\mathtt{D} \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{std}}, \mathtt{D}^{\mathtt{rrs}}\}$ *and for* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}\}$, *the proof systems* $\mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{D}, \mathtt{CubePol})$, $\mathtt{D}^{\mathtt{trv}} + \mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{D}, \mathtt{CubePol})$, *and* $\mathtt{D}^{\mathtt{std}} + \mathtt{QCDCL}^{\mathtt{ORD}}(\mathtt{D}, \mathtt{CubePol})$ *are equivalent to each other.*

We now introduce some new formulas which will be used to pinpoint the relative strengths of the proof systems.

The $\texttt{DoubleLongEq}_n$ formulas

The Equality formulas, first defined in [6], show that the proof system QCDCL with cube learning is stronger than QCDCL without [15]. We wish to show that cube-learning offers a similar advantage for systems with $\texttt{D}^{\texttt{rrs}}$. The Equality formulas cannot show this because $\texttt{D}^{\texttt{rrs}}(\texttt{Equality}) = \emptyset$; so using $\texttt{D}^{\texttt{rrs}}$ in any way makes them easy to refute irrespective of cube-learning. To achieve the desired separation, we modify the Equality formula by adding two clauses that make $\texttt{D}^{\texttt{rrs}}$ and $\texttt{D}^{\texttt{trv}}$ identical. These new formulas, called DoubleLongEq, maintain the separation without altering the hardness of Equality.

▶ **Formula 1.** *The* $\texttt{DoubleLongEq}_n$ *formula has the prefix* $\exists x_1 \cdots x_n \; \forall u_1 \cdots u_n \; \exists t_1 \cdots t_n$ *and the PCNF matrix*

$$\underbrace{(\bar{t}_1 \vee \cdots \vee \bar{t}_n)}_{T_n} \wedge \bigwedge_{i=1}^{n} \left[\underbrace{(x_i \vee u_i \vee t_i)}_{A_i} \wedge \underbrace{(\bar{x}_i \vee \bar{u}_i \vee t_i)}_{B_i} \right] \wedge$$
$$\underbrace{(\bar{u}_1 \vee \cdots \bar{u}_n \vee \bar{t}_1 \vee \cdots \bar{t}_n)}_{UT_n} \wedge \underbrace{(\bar{u}_1 \vee \cdots \bar{u}_n \vee t_1 \vee \cdots t_n)}_{UT'_n}$$

(Note: deleting the clauses UT_n and UT'_n gives the Equality formulas.)

▶ **Proposition 4.3.** *For* $\Phi = \texttt{DoubleLongEq}$, $\texttt{red}(\Phi) = \Phi$, *and* $\texttt{D}^{\texttt{rrs}}(\Phi) = \texttt{D}^{\texttt{std}}(\Phi) = \texttt{D}^{\texttt{trv}}(\Phi)$.

This makes the usage of the dependency schemes $\texttt{D}^{\texttt{rrs}}$ or $\texttt{D}^{\texttt{std}}$ completely useless. We now show that the formulas are easy to refute with cube-learning, but hard if cube-learning is switched off. Both these results closely mirror the corresponding results for Equality shown in [7] and [15] respectively.

▶ **Lemma 4.4.** *For* $\texttt{D}_1, \texttt{D}_2 \in \{\texttt{D}^{\texttt{trv}}, \texttt{D}^{\texttt{std}}, \texttt{D}^{\texttt{rrs}}\}$ *and* $\texttt{CubePol} \in \{\texttt{Cube-LD}, \texttt{Cube-D}_2\}$, *the* DoubleLongEq *formulas have polynomial size refutations in* $\texttt{D}_1 + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}_2, \texttt{CubePol})$.

▶ **Lemma 4.5.** *For* $\texttt{D}_1, \texttt{D}_2 \in \{\texttt{D}^{\texttt{trv}}, \texttt{D}^{\texttt{std}}, \texttt{D}^{\texttt{rrs}}\}$ *the* DoubleLongEq *formulas require exponential size refutations in* $\texttt{D}_1 + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}_2, \texttt{No-Cube})$.

Proof (Sketch). By the discussion above, it suffices to show that the formulas require exponential size refutations in $\texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{trv}}, \texttt{No-Cube})$.

In [13], the authors consider Σ^3 formulas with a specific structure, called XUT-formulas with the XT-property. They introduce a semantic measure called gauge for Σ^3 QBFs, and show that for an XUT-formula with the XT-property, the size of a refutation in $\texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{trv}}, \texttt{No-Cube})$ is at least exponential in its gauge.

The DoubleLongEq formulas are easily seen to be XUT-formulas with the XT-property. We show that they have linear gauge, implying that they are exponentially hard. ◀

The $\texttt{PreRRSTrapdoor}_n$ formulas

The next formula is designed to explore how adding $\texttt{D}^{\texttt{rrs}}$ in different ways affects the system. It sends QCDCL trails into a "trap" (of refuting the hard existential Pigeonhole Principle PHP; see Example 3.6) if $\texttt{D}^{\texttt{rrs}}$ is not used in propagation, but allows a short refutation (a contradiction on two variables) when $\texttt{D}^{\texttt{rrs}}$ is used. This leads to the definition of a formula inspired by the Trapdoor and Dep-Trap formulas from [7](Def. 4.5) and [16](Def. 4.4) respectively.

▶ **Formula 2.** *The* $\texttt{PreRRSTrapdoor}_n$ *formula has the prefix*
$\exists a\ \forall p\ \exists y_1, \cdots, y_{s_n}\ \forall w\ \forall v\ \exists t\ \exists x_1, \cdots, x_{s_n}\ \forall u\ \exists b\ \exists q\ \exists r\ \exists s$, *and the matrix is as given below.*

$$\begin{array}{ll} & \texttt{PHP}_n^{n+1}(x_1, \cdots, x_{s_n}) \\ \textit{for}\ \ i \in [s_n]: & (\bar{y}_i \vee x_i \vee u \vee b)\ ,\ (y_i \vee \bar{x}_i \vee u \vee b) \\ \textit{for}\ \ i \in [s_n]: & (y_i \vee w \vee v \vee t \vee b)\ ,\ (y_i \vee w \vee v \vee \bar{t} \vee b) \\ & (\bar{y}_i \vee w \vee v \vee t \vee b)\ ,\ (\bar{y}_i \vee w \vee v \vee \bar{t} \vee b) \\ \textit{for}\ \ i \in [s_n]: & (\bar{u} \vee \bar{b})\ ,\ (v \vee \bar{b} \vee \bar{r})\ ,\ (\bar{v} \vee b \vee s)\ ,\ (a \vee \bar{b})\ ,\ (\bar{a} \vee \bar{b})\ ,\ (p \vee q)\ ,\ (\bar{p} \vee \bar{q}) \end{array}$$

This formula has an unsatisfiable matrix (due to the presence of PHP).

The variable "w" is not necessary for the lower or upper bounds proved for this formula. Initialising $\texttt{PreRRSTrapdoor}|_{w=0}$ or removing the variable "w" entirely affects neither the bounds nor their proofs. However we keep it in because the PreRRSTrapdoor formulas are defined to extend the Trapdoor formula (defined in [7]) which has the "w" variable. Also, it shows that even if the preprocessing step (by $\texttt{D}^{\texttt{rrs}}$) is non-trivial and changes the formula, addition of $\texttt{D}^{\texttt{rrs}}$ in propagation can still make a difference.

▶ **Proposition 4.6.** $\texttt{D}^{\texttt{rrs}}(\texttt{PreRRSTrapdoor}) = \{(u, b), (v, b), (p, q)\}$.

Hence $\texttt{red-D}^{\texttt{rrs}}(\texttt{PreRRSTrapdoor}) = \texttt{PreRRSTrapdoor}|_{w=0}$. That is, if we preprocess using $\texttt{D}^{\texttt{rrs}}$, everything stays the same except that the variable w "disappears".

However, just preprocessing by $\texttt{D}^{\texttt{rrs}}$ is not enough to make this formula easy to refute. The following lemmas shows that the presence of $\texttt{D}^{\texttt{rrs}}$ *during propagation* is crucial to achieving polynomial sized refutations; its absence forces exponential size.

▶ **Lemma 4.7.** *For* $\texttt{CubePol} \in \{\texttt{No-Cube}, \texttt{Cube-LD}, \texttt{Cube-D}^{\texttt{rrs}}\}$ *the* PreRRSTrapdoor *formulas have polynomial size refutations in* $\texttt{D}^{\texttt{rrs}} + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{rrs}}, \texttt{CubePol})$

Proof (Sketch). By Observation 3.8, it suffices to show polynomial size refutations in $\texttt{D}^{\texttt{rrs}} + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{rrs}}, \texttt{No-Cube})$. Any trail must start with a decision on a. With $\texttt{D}^{\texttt{rrs}}$ in propagation, the contradiction on on y_1 and t is reached. Thus in 4 such trails the empty clause can be learnt, completing the refutation. ◀

▶ **Lemma 4.8.** *For* $\texttt{CubePol} \in \{\texttt{No-Cube}, \texttt{Cube-LD}, \texttt{Cube-D}^{\texttt{rrs}}\}$ *the* PreRRSTrapdoor *formulas require exponential size refutations in* $\texttt{D}^{\texttt{rrs}} + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{trv}}, \texttt{CubePol})$

Proof (Sketch). Since the PreRRSTrapdoor formulas have an unsatisfiable matrix, therefore, by Observation 3.9 it suffices to show hardness in $\texttt{D}^{\texttt{rrs}} + \texttt{QCDCL}^{\texttt{LEV-ORD}}(\texttt{D}^{\texttt{trv}}, \texttt{No-Cube})$.
Any trail starts with a decision on a, propagating $\bar{b}$, and then subsequently the trail falls into the trap of refuting Trapdoor and hence the PHP formulas; the hardness follows. ◀

The StdDepTrap formulas

The third new formula StdDepTrap we introduce shows the advantage of using $\texttt{D}^{\texttt{std}}$ in propagation and learning. The goal is to create clauses that can be learnt easily with $\texttt{D}^{\texttt{std}}$, but are hard to learn without it. These learned clauses enable a quick refutation in QCDCL with $\texttt{D}^{\texttt{std}}$, but without them, one is stuck refuting something hard.

▶ **Formula 3.** *The* $\texttt{StdDepTrap}_n$ *formula has the prefix*
$\exists b\ \forall w_1\ \exists z_1, \cdots, z_{s_n}\ \forall w_2\ \exists a\ \exists d\ \exists c\ \forall u\ \exists x\ \exists y\ \exists p\ \exists e_1\ \exists e_2$, *and the matrix is as given below.*

$$\begin{array}{cc} \bar{b} \vee \texttt{PHP}_n^{n+1}(z_1, \cdots, z_{s_n}) & \\ (y \vee p)\ ,\ (y \vee \bar{p}), & (w_1 \vee e_1)\ ,\ (w_2 \vee e_2) \\ (b \vee y)\ ,\ (a \vee \bar{y})\ ,\ (\bar{a} \vee x)\ ,\ (\bar{c} \vee u \vee \bar{x}), & (d \vee c \vee \bar{y})\ ,\ (\bar{d} \vee c \vee \bar{y}) \end{array}$$

The variables w_1, w_2, u are essentially "separators", putting existential variables into different levels. The variables e_1, e_2, x are blockers, ensuring that the separators do not get reduced too early in the trail.

▶ **Proposition 4.9.** $\mathtt{D}^{\mathtt{std}}(\mathtt{StdDepTrap}) = \{(w_1, e_1), (w_2, e_2), (u, x)\}$.

When $\mathtt{D}^{\mathtt{std}}$ is used in propagation, we show that these formulas have short refutations.

▶ **Lemma 4.10.** *For* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$ *the* StdDepTrap *formulas have polynomial size refutations in* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{CubePol})$

Proof (Sketch). By Observation 3.8, it suffices to show short refutations in the system $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{No\text{-}Cube})$. Consider the trail $\mathcal{T}_1 = \bar{\mathbf{b}}, y, a, x, \bar{c}, d, \square$ with learnable sequence $L_{\mathcal{T}_1} = \{c \vee \bar{y}, u \vee \bar{x} \vee \bar{y}, \bar{a} \vee \bar{y}, \bar{y}, b\}$. We choose to learn $\bar{y}$. We construct next the trail $\mathcal{T}_2 = \bar{y}, p, \square$ with learnable sequence $L_{\mathcal{T}_2} = \{y \vee \bar{p}, y, \square\}$. Thus the empty clause $\square$ is learnt. ◀

However, without $\mathtt{D}^{\mathtt{std}}$, the propagations force refuting the PHP clauses, which is hard.

▶ **Lemma 4.11.** *For* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{trv}}\}$ *the* StdDepTrap *formulas require exponential size refutations in* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol})$

Proof (Sketch). We first show that the the StdDepTrap formulas require exponential size without cube learning i.e. in $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ and then extend the argument for the case of $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ and $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}D}^{\mathtt{trv}})$

Every $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ trail must start with a decision on b, until a literal on b is learnt; thenceforth a trail must begin by propagating that literal. Any trail starting with b reaches a conflict in the PHP clauses, so subsequently exponential size is required. It can be shown that no satisfying trail is ever constructed, so cube-learning offers no advantage. ◀

Strength Relations between the Proof Systems

The hardness of formulas in QCDCL systems with or without cube-learning and dependency schemes are collected in Table 1, including known bounds as well as those established above.

Using these bounds, we now establish various relations between the newly introduced LEV-ORD-based proof systems and also between them and other QCDCL-based proof systems. Figure 1 summarises these and earlier known relations in a visually clear way.

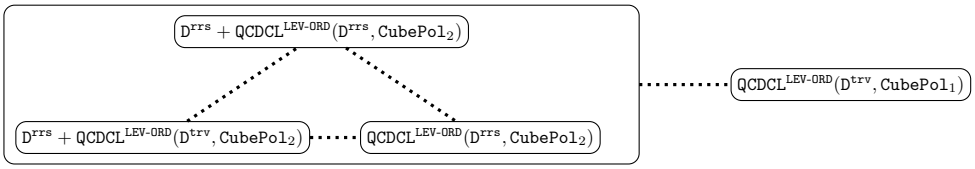
As we have discussed, cube-learning in QCDCL proof systems is the concept of allowing "terms" to be learnt from satisfying trails, and intuitively more (optional) learning power would mean a more powerful proof system. The first theorem validates this claim . It states that for any QCDCL proof system with dependency schemes, adding cube-learning yields a more powerful system than the corresponding system without it.

▶ **Theorem 4.12.** *For* $\mathtt{D}_1, \mathtt{D}_2 \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{std}}, \mathtt{D}^{\mathtt{rrs}}\}$ *and* $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}_2\}$ *the proof system* $\mathtt{D}_1 + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}_2, \mathtt{CubePol})$ *is strictly stronger than* $\mathtt{D}_1 + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}_2, \mathtt{No\text{-}Cube})$

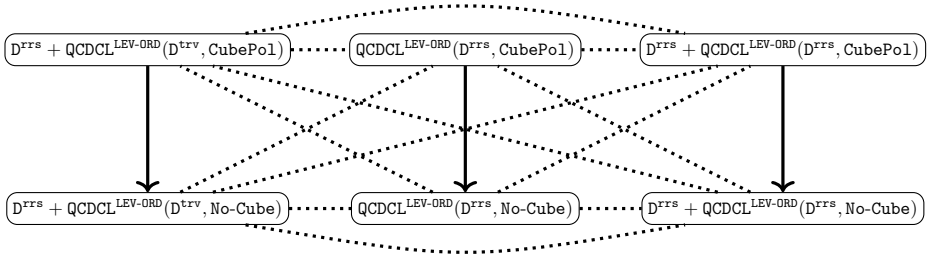
For the next three theorems we focus specifically on $\mathtt{D}^{\mathtt{rrs}}$. The first of these shows that irrespective of the manner in which $\mathtt{D}^{\mathtt{rrs}}$ is used in a QCDCL proof system, the resulting system is provably incomparable in strength to the QCDCL system that does not use dependency schemes. This was already known for the setting without cube-learning, Theorem 5 in [16]. We show here that cube learning makes no difference; the systems still remain incomparable. This highlights the fact that adding dependency schemes is not always beneficial.

Table 1 Formulas, their dependencies, and ease/hardness of refutations.
"$\mathtt{QCDCL}$ with $\mathtt{D}^{\mathtt{rrs}}$" includes $\{\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}, \mathtt{No\text{-}Cube})\}$ where $\mathtt{D} \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{rrs}}\}$, as well as $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$.
"$\mathtt{QCDCL}^{\mathtt{cube}}$ with $\mathtt{D}^{\mathtt{rrs}}$" includes $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}, \mathtt{CubePol})$ as well as $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{CubePol})$,
where $\mathtt{D} \in \{\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{rrs}}\}$ and $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$.

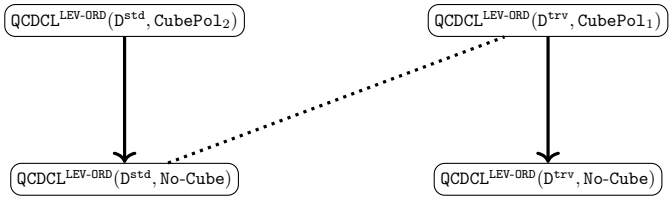
Formula	Matrix	Dependencies	Weakest System where Easy	Strongest System where Hard
`Equality` ([6]-Def. 3.1)	Sat	$\mathtt{D}^{\mathtt{rrs}} = \emptyset$ $\mathtt{D}^{\mathtt{std}} = \mathtt{D}^{\mathtt{trv}}$	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ [15]-Prop 5.2 $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}D}^{\mathtt{trv}})$ [15]-Prop 5.2 $\mathtt{QCDCL}$ with $\mathtt{D}^{\mathtt{rrs}}$ [16]-Lemma 1 $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{Cube\text{-}LD})$ $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}D}^{\mathtt{std}})$	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ [7]-Cor 5.8 $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{No\text{-}Cube})$
`TwinEq` ([15]-Def 6.1)	Sat	$\mathtt{D}^{\mathtt{rrs}} = \emptyset$ $\mathtt{D}^{\mathtt{std}} = \mathtt{D}^{\mathtt{trv}}$	$\mathtt{QCDCL}$ with $\mathtt{D}^{\mathtt{rrs}}$ [16]-Sec 4.9	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ [15]-Prop 6.3
`Trapdoor` ([7]-Def 4.5)	Unsat	$\mathtt{D}^{\mathtt{rrs}} = \emptyset$ $\mathtt{D}^{\mathtt{std}} = \{(w, t)\}$	$\mathtt{QCDCL}$ with $\mathtt{D}^{\mathtt{rrs}}$ [16]-Lemma 2	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ (Obs. 3.9,) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}D}^{\mathtt{trv}})$ (and [7]-Prop 4.6) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{Cube\text{-}LD})$ $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{Cube\text{-}D}^{\mathtt{std}})$
`Dep-Trap` ([16]-Sec 4.4)	Unsat	$\mathtt{D}^{\mathtt{rrs}} = \{(w, t)\}$ [16] $\mathtt{D}^{\mathtt{std}} = \{(w, t)\} \cup \{(u_i, x_i)\}$	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ [16]-Lemma 3 $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{No\text{-}Cube})$	$\mathtt{QCDCL}^{\mathtt{cube}}$ with $\mathtt{D}^{\mathtt{rrs}}$ ([16], Obs. 3.9)
`TwoPHPandCT` ([16]-Sec 4.5)	Unsat	$\mathtt{D}^{\mathtt{rrs}} = \emptyset$ [16]	$\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ [16]-Lemma 5 $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$ [16]	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ ([16],Obs. 3.9) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}D}^{\mathtt{trv}})$ ([16],Obs. 3.9) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}LD})$ ([16],Obs. 3.9) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}})$ ([16],Obs. 3.9)
`PreDepTrap` ([16]-Sec 4.7)	Sat $\mathtt{red\text{-}D}^{\mathtt{rrs}}$: Unsat	$\mathtt{D}^{\mathtt{rrs}} = \{(w, t)\}$ [16]	$\mathtt{QCDCL}$[16] $\mathtt{QCDCL}(\mathtt{D}^{\mathtt{rrs}})$[16]	$\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{cube}}$([16],Obs. 3.9) $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{cube}}(\mathtt{D}^{\mathtt{rrs}})$([16],Obs. 3.9)
`PropDep-Trap` ([16]-Sec 4.8)	Unsat	$\mathtt{D}^{\mathtt{rrs}} = \{(w, t), (b_1, z_1), (b_2, z_2)\}$ [16]	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ ([16]) $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ ([16])	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}LD})$ ([16], Obs. 3.9) $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}})$ ([16], Obs. 3.9) $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}LD})$ ([16], Obs. 3.9) $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}})$ ([16], Obs. 3.9)
*`DoubleLongEq`	Sat	$\mathtt{D}^{\mathtt{rrs}} = \mathtt{D}^{\mathtt{trv}}$ (Proposition 4.3)	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ (Lemma 4.4) $\mathtt{QCDCL}^{\mathtt{cube}}$ with $\mathtt{D}^{\mathtt{rrs}}$ (Lemma 4.4)	$\mathtt{QCDCL}$ (Lemma 4.5) $\mathtt{QCDCL}$ with $\mathtt{D}^{\mathtt{rrs}}$ (Lemma 4.5)
*`PreRRSTrapdoor`	Unsat	$\mathtt{D}^{\mathtt{rrs}} = \{(u, b), (v, b), (p, q)\}$ (Proposition 4.6)	$\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$ (Lemma 4.7)	$\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ (Lemma 4.8)
*`StdDepTrap`	Sat	$\mathtt{D}^{\mathtt{std}} = \{(u, x), (w_1, e_1), (w_2, e_2)\}$ (Proposition 4.9)	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{No\text{-}Cube})$ (Lemma 4.10)	$\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{Cube\text{-}LD})$ (Lemma 4.11)



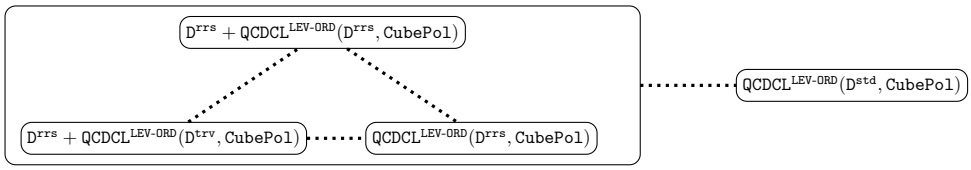
(a) For $\mathtt{CubePol}_1 \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{trv}}\}$; $\mathtt{CubePol}_2 \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$.
From Theorems 4.13 and 4.14.



(b) For $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$.
From Theorems 4.12 and 4.15.



(c) For $\mathtt{CubePol}_1 \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{trv}}\}$, $\mathtt{CubePol}_2 \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{std}}\}$.
From Theorems 4.12 and 4.16.



(d) For $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{std}}\}$. From Theorem 4.17.

Figure 1 The simulation order of various QCDCL systems.
$A \to B$ means A simulates B but B does not simulate A.
$A \cdots B$ means neither A nor B simulates the other.

▶ **Theorem 4.13.** *Any proof systems* $\mathtt{P}_1, \mathtt{P}_2$ *are incomparable, where*

$$\mathtt{P}_1 \in \{\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol}) \mid \mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}\} \text{ and}$$

$$\mathtt{P}_2 \in \left\{ \mathtt{D}_1 + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}_2, \mathtt{CubePol}) \mid \begin{array}{l} (\mathtt{D}_1, \mathtt{D}_2) \in \{(\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{rrs}}), (\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{trv}}), (\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{rrs}})\}, \\ \mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}_2\} \end{array} \right\}.$$

The next theorem shows that adding $\mathtt{D}^{\mathtt{rrs}}$ in different ways to QCDCL yields systems incomparable in strength. (Again, this was already known for the setting without cube-learning, Theorem 4 in [16].) This highlights that adding dependency schemes in any one of preprocessing or propagation and learning is not inherently better than the other.

▶ **Theorem 4.14.** *For a fixed* $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$*, the three systems* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{CubePol})$*,* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol})$*, and* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{CubePol})$ *are pairwise incomparable.*

Earlier, we have seen that adding cube-learning to a QCDCL system with dependency always yields a stronger system, Theorem 4.12. The following theorem shows that even adding cube-learning to a QCDCL system using $\mathtt{D}^{\mathtt{rrs}}$ is incomparable to a QCDCL system without cube-learning but using $\mathtt{D}^{\mathtt{rrs}}$ in a different way.

▶ **Theorem 4.15.** *For any* $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{rrs}}\}$*, adding* $\mathtt{D}^{\mathtt{rrs}}$ *to the proof system* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol})$ *in one way and to* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ *in any different way yields incomparable proof systems. In particular,*

1. $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol})$ *is incomparable with* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$ *and* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$.
2. $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{CubePol})$ *is incomparable with* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$ *and* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$.
3. $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{CubePol})$ *incomparable with* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{rrs}}, \mathtt{No\text{-}Cube})$ *and* $\mathtt{D}^{\mathtt{rrs}} + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{No\text{-}Cube})$.

Now we come to $\mathtt{D}^{\mathtt{std}}$. First, we show that, as for $\mathtt{D}^{\mathtt{rrs}}$, a QCDCL system with $\mathtt{D}^{\mathtt{std}}$ is incomparable in strength to a standard QCDCL system without any dependency schemes.

▶ **Theorem 4.16.** *For* $\mathtt{CubePol} \in \{\mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{std}}\}$*, the proof systems* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{No\text{-}Cube})$ *and* $\mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{trv}}, \mathtt{CubePol})$ *are incomparable.*

Finally, we compare the dependency schemes $\mathtt{D}^{\mathtt{rrs}}$ and $\mathtt{D}^{\mathtt{std}}$. The mere fact that $\mathtt{D}^{\mathtt{rrs}}$ is a refinement of $\mathtt{D}^{\mathtt{std}}$ (more general, eliminates more dependencies) does not make it better; for that matter, $\mathtt{D}^{\mathtt{rrs}}$ is a refinement of $\mathtt{D}^{\mathtt{trv}}$, but we have seen that using it can be a disadvantage for some formulas. Similarly, we prove below that neither of $\mathtt{D}^{\mathtt{rrs}}$ and $\mathtt{D}^{\mathtt{std}}$ has a proof-theoretic advantage over the other, irrespective of the presence or absence of cube-learning.

▶ **Theorem 4.17.** *For any* $(\mathtt{D}_1, \mathtt{D}_2) \in \{(\mathtt{D}^{\mathtt{trv}}, \mathtt{D}^{\mathtt{rrs}}), (\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{trv}}), (\mathtt{D}^{\mathtt{rrs}}, \mathtt{D}^{\mathtt{rrs}})\}$*, and* $\mathtt{CubePol} \in \{\mathtt{No\text{-}Cube}, \mathtt{Cube\text{-}LD}, \mathtt{Cube\text{-}D}^{\mathtt{std}}\}$*, the proof systems* $\mathtt{P}_1 \in \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}^{\mathtt{std}}, \mathtt{CubePol})$ *and* $\mathtt{P}_2 \in \mathtt{D}_1 + \mathtt{QCDCL}^{\mathtt{LEV\text{-}ORD}}(\mathtt{D}_2, \mathtt{CubePol})$ *are incomparable.*

5 Conclusions

In the context of QBF proof systems, dependency schemes are expected to aid the process of refutation, as indeed they do for Q-Res and QU-Res. It was thus a surprise to see this advantage does not automatically translate to QCDCL algorithms (even in the presence

of cube learning) if they make only level-ordered decisions. However, we lack proper lower bound techniques for a non-trivial decision policy D-ORD. Even for the level-ordered policy, there are many unresolved questions. The comparison between $\mathtt{D}^{\mathtt{trv}}$ and $\mathtt{D}^{\mathtt{rrs}}$ already shows that using a more refined scheme is not necessarily an advantage. The relative strengths of $\mathtt{D}^{\mathtt{std}}$ and $\mathtt{D}^{\mathtt{rrs}}$ (which refines $\mathtt{D}^{\mathtt{std}}$) is not yet clear. How other dependency schemes would relate in this scenario is completely unexplored.

Since QCDCL-style reasoning is explained through long-distance clause/term resolution, it is worth highlighting three long-standing open questions about those systems. (1) Does using dependency schemes confer any advantage with clausal long-distance; i.e. is the simulation of LDQ-Res by LDQ(D)-Res strict for $\mathtt{D}^{\mathtt{std}}$ or $\mathtt{D}^{\mathtt{rrs}}$? (2) Is the use of dependency schemes in long-distance term resolution (the system LDQ-TermRes) sound? (3) Can we even show that $\mathtt{Q}(\mathtt{D}^{\mathtt{std}})$-Res is strictly more powerful than Q-Res? This question has remained open since $\mathtt{D}^{\mathtt{std}}$ was first implemented in DepQBF over 15 years ago.

References

1. Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi:10.1613/jair.3152.
2. Salman Azhar, Gary Peterson, and John Reif. Lower bounds for multiplayer non-cooperative games of incomplete information. *Journal of Computers and Mathematics with Applications*, 41:957–992, 2001.
3. Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.
4. Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 2004. doi:10.1613/jair.1410.
5. Olaf Beyersdorff. Proof complexity of quantified Boolean logic – a survey. In Marco Benini, Olaf Beyersdorff, Michael Rathjen, and Peter Schuster, editors, *Mathematics for Computation (M4C)*, pages 353–391. World Scientific, 2022.
6. Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:13)2019.
7. Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. *Logical Methods in Computer Science*, 19(2), 2023. preliminary version in ITCS 2021. doi:10.46298/lmcs-19(2:2)2023.
8. Olaf Beyersdorff, Benjamin Böhm, and Meena Mahajan. Runtime vs. extracted proof size: an exponential gap for CDCL on QBFs. In *Proceedings of 38th AAAI Conference on Artificial Intelligence, 2024*, 2024.
9. Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Trans. Comput. Theory*, 11(4):26:1–26:42, 2019. doi:10.1145/3352155.
10. Joshua Blinkhorn and Olaf Beyersdorff. Shortening QBF proofs with dependency schemes. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2017. doi:10.1007/978-3-319-66263-3_17.
11. Joshua Blinkhorn and Olaf Beyersdorff. Dynamic dependency awareness for QBF. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5224–5228. ijcai.org, 2018. doi:10.24963/ijcai.2018/726.

12 Joshua Blinkhorn, Tomás Peitl, and Friedrich Slivovsky. Davis and Putnam meet Henkin: Solving DQBF with resolution. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2021. doi:10.1007/978-3-030-80223-3_4.

13 Benjamin Böhm and Olaf Beyersdorff. Lower bounds for QCDCL via formula gauge. *J. Autom. Reason.*, 67(4):35, 2023. preliminary version in SAT 2021. doi:10.1007/s10817-023-09683-1.

14 Benjamin Böhm and Olaf Beyersdorff. QCDCL vs QBF resolution: Further insights. *J. Artif. Intell. Res.*, 81:741–769, 2024. preliminary version in SAT 2023. doi:10.1613/jair.1.15522.

15 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. QCDCL with cube learning or pure literal elimination - what is best? *Artif. Intell.*, 336:104194, 2024. preliminary version on IJCAI 2022. doi:10.1016/j.artint.2024.104194.

16 Abhimanyu Choudhury and Meena Mahajan. Dependency schemes in CDCL-based QBF solving: A proof-theoretic study. *J. Autom. Reason.*, 68(3):16, 2024. preliminary version in FSTTCS 2023. doi:10.1007/s10817-024-09707-4.

17 Abhimanyu Choudhury and Meena Mahajan. On the interplay of cube learning and dependency schemes in QCDCL proof systems, 2025. doi:10.48550/arXiv.2510.05876.

18 Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 761–780. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-761.

19 Florian Lonsing. *Dependency schemes and search-based QBF solving: theory and practice*. PhD thesis, Johannes Kepler University, Linz, Austria, 2012.

20 Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *J. Satisf. Boolean Model. Comput.*, 7(2-3):71–76, 2010. doi:10.3233/sat190077.

21 Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proceedings of International Conference on Automated Deduction (CADE)*, pages 371–384, 2017. doi:10.1007/978-3-319-63046-5_23.

22 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021. doi:10.3233/FAIA200987.

23 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *J. Artif. Intell. Res.*, 65:180–208, 2019. doi:10.1613/jair.1.11529.

24 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Long-distance Q-resolution with dependency schemes. *J. Autom. Reasoning*, 63(1):127–155, 2019. doi:10.1007/s10817-018-9467-3.

25 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

26 Marko Samer and Stefan Szeider. Backdoor sets of quantified boolean formulas. *J. Autom. Reasoning*, 42(1):77–97, 2009. doi:10.1007/s10817-008-9114-5.

27 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *Proc. IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84, 2019. doi:10.1109/ICTAI.2019.00020.

28 João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.

29 Friedrich Slivovsky and Stefan Szeider. Soundness of Q-resolution with dependency schemes. *Theor. Comput. Sci.*, 612:83–101, 2016. doi:10.1016/j.tcs.2015.10.020.

30 Moshe Y. Vardi. Boolean satisfiability: theory and engineering. *Communications of the ACM*, 57(3):5, 2014. doi:10.1145/2578043.

31 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002. doi:10.1145/774572.774637.

Characterizing NC^1 with Typed Monoids

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK

Aidan T. Evans[1]
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

Krebs et al. (2007) gave a characterization of the complexity class TC^0 as the class of languages recognized by a certain class of typed monoids. The notion of typed monoid was introduced to extend methods of algebraic automata theory to infinite monoids and hence characterize classes beyond the regular languages. We advance this line of work beyond TC^0 by giving a characterization of NC^1. This is obtained by first showing that NC^1 can be defined as the languages expressible in an extension of first-order logic using only unary quantifiers over regular languages. The expressibility result is a consequence of a general result showing that finite monoid multiplication quantifiers of higher dimension can be replaced with unary quantifiers in the context of interpretations over strings, which also answers a question of Lautemann et al. (2001). We estblish this collapse result for a much more general class of interpretations using results on interpretations due to Bojańczyk et al. (2019), which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Complexity theory and logic; Theory of computation → Circuit complexity; Theory of computation → Finite Model Theory

Keywords and phrases algebraic automata theory, circuit complexity, descriptive complexity, typed monoids, semigroups, generalized quantifiers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.26

Related Version *Full Version*: `https://arxiv.org/abs/2508.11019` [8]

Funding *Anuj Dawar*: Research funded in part by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee: grant number EP/X028259/1.
Aidan T. Evans: Researched supported by Huawei HiSilicon Studentship.

1 Introduction

Much work in theoretical computer science is concerned with studying classes of formal languages, whether these are classes defined in terms of grammars and expressions, such as the class of regular or context-free languages, or whether they are *complexity classes* such as P and NP, defined by resource bounds on machine models. Indeed, the distinction between these is largely historical as most classes of interest admit different characterizations based on machine models, grammars, logical definability, or algebraic expressions. The class of regular languages can be characterized as the languages accepted by linear-time-bounded single-tape Turing machines [12] while P can be characterized without reference to resources as the languages recognized by multi-head two-way pushdown automata [7]. The advantage of the variety of characterizations is, of course, the fact that these bring with them different mathematical toolkits that can be brought to the study of the classes.

The class of regular languages has arguably the richest theory in this sense of diversity of characterizations. Most students of computer science learn of the equivalence of deterministic and nondeterministic finite automata, regular languages and linear grammars and many

[1] Corresponding Author

© Anuj Dawar and Aidan T. Evans;
licensed under Creative Commons License CC-BY 4.0

also know that the regular languages are exactly those definable in monadic second-order logic with an order predicate. Perhaps the most productive approach to the study of regular languages is via their connection to finite monoids. Every language L has a syntactic monoid, which is finite if, and only if, L is regular. Moreover, closure properties of classes of regular languages relate to natural closure properties of classes of monoids, via Eilenberg's Correspondence Theorem [11]. Finally, we have *Krohn-Rhodes theory* which decomposes regular languages into elementary components; such a decomposition has found applications not only in semigroup theory but also, for example, in physics [17, Sections 4.8a, 4.10] and the modeling of biochemical reactions [17, Chapter 6, Part I]. All together, these tools give rise to *algebraic automata theory* – which leads to the definition of natural subclasses of the class of regular languages, to effective decision procedures for automata recognizing such classes, and to separation results.

When it comes to studying computational complexity, we are mainly interested in classes of languages richer than just the regular languages. Thus, the syntactic monoids of the languages are not necessarily finite and the extensive tools of Krohn-Rhodes theory are not available to study them. Nonetheless, some attempts have been made to extend the methods of algebraic automata theory to classes beyond the regular languages. Most significant is the work of Krebs and collaborators [2, 3, 14, 13, 6], which introduces the notion of *typed monoids*. The idea is to allow for languages with infinite syntactic monoids, but limit the languages they recognize by associating with the monoids a finite collection of types. This allows for the formulation of a version of Eilenberg's Correspondence theorem associating closure properties on classes of typed monoids with corresponding closure properties of classes of languages. In particular, this implies that most complexity classes of interest can be uniquely characterized in terms of an associated class of typed monoids [3]. An explicit description of the class characterizing uniform TC^0 is given in Krebs et al. [14, 13] and, furthermore, provides a decomposition of TC^0 into elementary components analogous to the Krohn-Rhodes theory for regular languages. This is obtained through a general method which allows us to construct typed monoids corresponding to *unary quantifiers* defined from specific languages [13] (see also Theorem 11 below).

In this paper, we extend this work to obtain a characterization and decomposition of (uniform) NC^1 as the class of languages recognized by the collection of typed monoids obtained as the closure under *ordered strong block products* of three typed monoids: the group of integers with types for positive and non-positive integers; the monoid of the natural numbers with types for the square numbers and non-square numbers; and a finite non-solvable group such as S_5 with a type for each subset of the group. Full definitions of these terms follow below. Our result is obtained by first characterizing NC^1 in terms of logical definability in an extension of first-order logic with only unary quantifiers. It is known that any regular language whose syntactic monoid is a non-solvable group is complete for NC^1 under reductions definable in first-order logic with arithmetic predicates ($(\mathrm{FO})[+,\times]$) [1]. From this, we know we can describe NC^1 as the class of languages definable in an extension of $(\mathrm{FO})[+,\times]$ with quantifiers (of arbitrary arity) associated with the regular languages recognized by the monoid S_5. We show that the family of such quantifiers associated with any finite monoid can be replaced with just the unary quantifiers. This answers a question left open in Lautemann et al. [16] and allows us to obtain the sought after algebraic characterization.

For the purpose of establishing the characterization of NC^1, it suffices to consider quantifiers applied to interpretations in which tuples of elements are ordered lexicographically. However, we show that the arity collapse of quantifiers associated with finite monoids holds more generally, for any first-order definable linear orders on tuples. To show this, we leverage a characterization of first-order interpretations due to Bojańczyk et al. [5, 4].

We begin in Section 2, by covering relevant background on semigroup theory, typed monoids, and multiplication quantifiers (some of which is relegated to the appendix in the interests of space). In Section 3, we establish the technical result showing that quantifiers of higher arity over a regular language L can be defined using just unary quantifiers over the syntactic monoid of L, when the quantifiers are applied to interpretations with a lexicographic order on tuples. In Section 4, we apply this to obtain the algebraic characterization of NC^1. finally, in Section 5, we establish the arity collapse for quantifiers in the context of more general interpretations. A full version of this paper with a detailed background is available on arXiv [8].

2 Preliminaries

We assume the reader is familiar with basic concepts of formal language theory, automata theory, complexity theory, and logic. We quickly review definitions we need in order to fix notation and establish conventions.

We write $\mathbb{Z}$ for the set of integers, $\mathbb{N}$ for the set of natural numbers (including 0), and $\mathbb{Z}^+$ for the set of positive integers. We write $[n]$ for the set of integers $\{1, \ldots, n\}$ and $\mathbb{S}$ for the set of *square* integers. That is, $\mathbb{S} = \{x \in \mathbb{Z}^+ \mid x = y^2 \text{ for some } y \in \mathbb{Z}\}$.

For a fixed $n \in \mathbb{Z}^+$ and an integer $i \in [n]$, we define the *n-bit one-hot encoding* of i to be the binary string $b \in \{0,1\}^n$ such that $b_j = 1$ if, and only if, $j = i$. For any set S, we write $\wp(S)$ for the powerset of S.

The complexity classes TC^0 and NC^1 are classes of languages defined in terms of circuits. We are only interested in the *uniform* versions of these classes. Specifically, TC^0 is the class of languages recognized by a uniform family of circuits of polynomial size and constant depth using And, Or, Not and Majority gates of arbitrary fan-in; and NC^1 is the class of languages recognized by a uniform family of circuits of polynomial size and logarithmic depth using And, Or and Not gates of fan-in at most 2. Since gates (including majority gates) of unbounded fan-in can be simulated by circuits of logarithmic depth of fan-in 2 using just the standard Boolean basis, we have $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$ and the separation of the two classes is open. The requirement of *uniformity* here means that the circuits of input size n are easily computed from n. It is standard to take "easily computed" to mean DLogTime-uniform, though the classes are robust under varying the definition (see [20]).

2.1 Semigroups, Monoids, and Groups

A *semigroup* $(S, \cdot)$ is a set S equipped with an *associative* binary operation. We call a semigroup *finite* if S is finite. Context permitting, we may refer to a semigroup $(S, \cdot)$ simply by its underlying set S.

A *monoid* $(M, \cdot)$ is a semigroup with a distinguished element $1_M \in M$ such that for all $m \in M$, $1_M \cdot m = m \cdot 1_M = m$. We call 1_M the *identity* or *neutral* element of M. A *group* $(G, \cdot)$ is a monoid such that for every $g \in G$, there exists an element $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = 1$. We call g^{-1} the *inverse* of g.

Note that $(\mathbb{Z}, +)$ is a group, $(\mathbb{N}, +)$ is a monoid but not a group and $(\mathbb{Z}^+, +)$ is a semigroup but not a monoid. In the first two cases, the identity element is 0. When we refer to the monoids $\mathbb{Z}$ or $\mathbb{N}$ we assume that the operation referred to is standard addition.

For a semigroup $(S, \cdot)$, we say that a set $G \subseteq S$ *generates* S if S is equal to the closure of G under $\cdot$; we denote this by $S = \langle G \rangle$., or, simply, $\langle G \rangle$ if the operation is clear from context, and call G a *generating set of* S. We say that S is *finitely generated* if there exists a finite generating set of S. All semigroups we consider are finitely generated. Note that $\mathbb{Z}^+$ is generated by $\{1\}$, $\mathbb{N}$ by $\{0, 1\}$ and $\mathbb{Z}$ by $\{1, -1\}$.

We write U_1 for the monoid $(\{0,1\},\cdot)$ where the binary operation is the standard multiplication. Note that 1 is the identity element here. For any set S, we denote by S^+ the set of non-empty finite strings over S and by S^* the set of all finite strings over S. Equipped with the concatenation operation on strings, which we denote by either $\circ$ or simply juxtaposition, S^* is a monoid and S^+ is a semigroup but not a monoid. We refer to these as the *free monoid* and *free semigroup* over S, respectively. Note that S is a set of generators for S^+ and $S \cup \{\epsilon\}$ is a set of generators for S^*.

A monoid homomorphism from a monoid $(S,\cdot_S)$ to a monoid $(T,\cdot_T)$ is a function $h : S \to T$ such that for all $s_1, s_2 \in S$, $h(s_1 \cdot_S s_2) = h(s_1) \cdot_T h(s_2)$ and $h(1_S) = 1_T$. A *congruence* on a monoid $(M,\cdot)$ is an equivalence relation $\sim$ on M such that for all $a,b,c,d \in M$, if $a \sim b$ and $c \sim d$, then $a \cdot c \sim b \cdot d$. We denote by $M/\!\sim$ the set of equivalence classes of $\sim$ on M. We denote by $[a]_\sim$, or simply $[a]$, the equivalence class of $a \in M$ under $\sim$. Any congruence $\sim$ gives rise to the *quotient monoid* of M by $\sim$, namely the monoid $(M/\!\sim,\star)$ where for $[a],[b] \in M/\!\sim$, $[a] \star [b] = [a \cdot b]$. The map $\eta : M \to M/\!\sim$ defined by $\eta(a) = [a]$ is then a homomorphism, known as the *canonical homomorphism* of M onto $M/\!\sim$.

2.2 Language Recognition with Monoids

Here we recall the definition of syntactic congruences and syntactic monoids.

For a language $L \subseteq \Sigma^*$ over an alphabet Σ, we define the *syntactic congruence* of L as the equivalence relation $\sim_L$ on Σ^* such that for all $x,y \in \Sigma^*$, $x \sim_L y$ if, and only if, for all $w,v \in \Sigma^*$, $wxv \in L$ if, and only if, $wyv \in L$.

It is easily seen that this relation is a congruence on the free monoid Σ^*. The quotient monoid $\Sigma^*/\!\sim_L$ is known as the *syntactic monoid* of L. More generally, we say that a monoid M *recognizes* a language L if there is a homomorphism $h : \Sigma^* \to M$ and a set $A \subseteq M$ such that $L = h^{-1}(A)$. It is easily seen that the syntactic monoid of L recognizes L. A language is regular if, and only if, its syntactic monoid is finite [19].

2.3 Logics and Quantifiers

We assume familiarity with the basic syntax and semantics of first-order logic. In this paper, the logic is always interpreted in finite relational structures. We generally denote structures by Fraktur letters, $\mathfrak{A}$, $\mathfrak{B}$, etc., and the corresponding universe of the structure is denoted $|\mathfrak{A}|$, $|\mathfrak{B}|$, etc. We are almost exclusively interested in *strings* over a finite alphabet. Thus, fix an alphabet Σ. A Σ-string is then a structure $\mathfrak{A}$ whose universe $|\mathfrak{A}|$ is linearly ordered by a binary relation $<$ and which interprets a set of unary relation symbols $(R_\sigma)_{\sigma\in\Sigma}$ such that for each element $a \in |\mathfrak{A}|$ there is a unique $\sigma \in \Sigma$ such that a is in the interpretation of R_σ.

More generally, let τ be any relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1,\ldots,R_k$. We can associate with any τ-structure in which $<$ is a linear order a string over an alphabet of size 2^k as formalized in the following definition.

▶ **Definition 1.** *For τ a relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1,\ldots,R_k$, and $\mathfrak{A}$ a τ-structure with n elements that interprets the symbol $<$ as a linear order of its universe, we define the string $w_\mathfrak{A}$* associated with *$\mathfrak{A}$ as the string of length n over the alphabet $\Sigma_k = \{0,1\}^k$ of size 2^k so that if a is the ith element of $w_\mathfrak{A}$, then a is the k-tuple where $a_j = 1$ if, and only if, R_j holds at the ith element of $\mathfrak{A}$.*

For example, say $\tau = \{R_1,R_2,R_3\}$ and $\mathfrak{A} = ([4],<^\mathfrak{A},R_1^\mathfrak{A},R_2^\mathfrak{A},R_3^\mathfrak{A})$ where $1 < 2 < 3 < 4$, $R_1^\mathfrak{A} = \{1,3\}$, $R_2^\mathfrak{A} = \{2,4\}$, and $R_3^\mathfrak{A} = \{1,2,3\}$. Then, the string $w_\mathfrak{A} = (101)(011)(101)(010)$.

Using the above, we can associate a language with any isomorphism-closed class of structures over the vocabulary τ. We formalize this definition for future use.

▶ **Definition 2.** *For τ a relational vocabulary consisting of a binary relation symbol $<$ and unary relation symbols $R_1, \ldots, R_k$, and $\mathcal{A}$ a class of τ-structures where every structure interprets $<$ as a linear order, we define the language $L_{\mathcal{A}}$ over the alphabet $\Sigma_k = \{0,1\}^k$ to be*

$$L_{\mathcal{A}} = \{w_{\mathfrak{A}} \mid \mathfrak{A} \in \mathcal{A}\}.$$

Conversely, for any language L over the alphabet Σ_k, we define the class of τ-structures $\mathcal{S}_L$ to be

$$\mathcal{S}_L = \{\mathfrak{A} \mid w_{\mathfrak{A}} \in L\}.$$

It is worth noting that the above operations are not inverses, in the sense that it is not the case that $S_{L_{\mathcal{A}}} = \mathcal{A}$. This is because while $\mathcal{A}$ is a class of structures using k unary relation symbols, $S_{L_{\mathcal{A}}}$ is in a vocabulary using 2^k unary symbols.

As the elements of a string $\mathfrak{A}$ are linearly ordered, we can identify them with an initial segment $\{1, \ldots, n\}$ of the positive integers. In other words, we treat a string with universe $\{1, \ldots, n\}$ and the standard order on these elements as a canonical representative of its isomorphism class. In addition to the order predicate, we may allow other *numerical predicates* to appear in formulas of our logics. These are predicates whose meaning is completely determined by the size n of the structure and the ordering of its elements. In particular, we have ternary predicates $+$ and $\times$ for the partial addition and multiplication functions.

An insight due to Lindström allows us to define a *quantifier* from any isomorphism-closed class of structures (see [10]). We adopt the terminology of Ebbginhaus and Flum [9, Chapter 12] and provide a brief overview now. Consider a relational vocabulary $\tau = \{R_1, \ldots, R_l\}$, where for each i, R_i is a relation symbol of arity r_i. For any vocabulary σ and positive integer d, an *interpretation* of τ in σ of dimension d is a tuple of formulas $I = (\phi_1(\overline{x}_1), \ldots, \phi_l(\overline{x}_l))$ of vocabulary σ where ϕ_i is associated with a tuple $\overline{x}_i$ of variables of length $|\overline{x}_i| = dr_i$. Suppose we are given a σ-structure $\mathfrak{A}$ and an assignment α that takes variables to elements of $\mathfrak{A}$. Then let $\phi_i^{\mathfrak{A},\alpha}$ denote the relation of arity dr_i consisting of the set of tuples $\{\overline{a} \in |\mathfrak{A}|^{dr_i} \mid \mathfrak{A} \models \phi_i[\alpha[\overline{x}_i/\overline{a}]]\}$. Then, the interpretation I defines a map that takes a σ-structure $\mathfrak{A}$, along with an assignment α to the τ-structure $I(\mathfrak{A}, \alpha)$ with universe $|\mathfrak{A}|^d$ where the interpretation of R_i is the set $\phi_i^{\mathfrak{A},\alpha}$, seen as a relation of arity r_i on $|\mathfrak{A}|^d$.

Given a class of τ-structures Q and any positive integer d, we have a quantifier Q_d. In a logic with Q_d, we can form formulas of the form

$$Q_d \overline{x}_1 \cdots \overline{x}_l(\phi_1, \ldots, \phi_l)$$

whenever $I = (\phi_1(\overline{x}_1), \ldots, \phi_l(\overline{x}_l))$ is an interpretation of dimension d. In this formula, occurrences in the subformula ϕ_i of variables among x_i are bound. The semantics of this quantifier are given by the rule that $Q_d \overline{x}_1 \cdots \overline{x}_l(\phi_1, \ldots, \phi_l)$ is true in a structure $\mathfrak{A}$ under some assignment α of values to the free variables if the τ-structure $I(\mathfrak{A}, \alpha)$ is in Q. We can understand Q_d as the dth vectorization of the quantifier Q (see [9, Def. 12.1.6]). When $d = 1$, we may omit the subscript. A quantifier of the form Q_1 is called *unary*.

The standard first-order quantifiers: $\exists$ and $\forall$ can be seen as special cases of Lindström quantifiers in a vocabulary with one unary relation U. The existential quantifier consists of all structures (A, U) where $U \subseteq A$ is non-empty and the universal quantifier consists of all structures (A, U) where $U = A$.

We are particularly interested in interpretations I where both σ and τ are vocabularies of strings. These are also known in the literature as *string-to-string transducers*. (See [4] for an example of how transducers may have many representations.) Therefore, I must define an interpretation of not only each $R_i \in \tau$ (using the formula ϕ_i) but also a σ-formula $\phi_<$ defining the linear order on the universe of the τ-structure $I(\mathfrak{A}, \alpha)$. We are particularly interested in the case where the order defined is the lexicographic order on d-tuples of $|\mathfrak{A}|$ induced by the order in $\mathfrak{A}$. This order is easily defined by a (quantifier-free) first-order formula, and we often omit it from the description of I.

When we consider interpretations that define a linear order other than the lexicographic order, we explicitly include the formula defining the order and thus the formula is $Q_d \overline{x}\overline{y}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \phi_l(\overline{y}))$, where the interpretation is of dimension d and so $|\overline{x}| = 2d$ and $|\overline{y}| = d$.

Finally, we now introduce some notation we use in the rest of the paper for various logics formed by combining particular choices of quantifiers and numerical predicates.

▶ **Definition 3.** *For a set of quantifiers $\mathfrak{Q}$ and numerical predicates $\mathfrak{N}$, we denote by $(\mathfrak{Q})[\mathfrak{N}]$ the logic constructed by extending quantifier-free first-order logic with the quantifiers in $\mathfrak{Q}$ and allowing the numerical predicates in $\mathfrak{N}$.*

We denote by FO *the set of standard first-order quantifiers: $\{\exists, \forall\}$.*

When $\mathfrak{Q}$ is just a singleton $\{Q\}$, we sometimes denote $(\mathfrak{Q})[\mathfrak{N}]$ by $(Q)[\mathfrak{N}]$ or $(Q_1)[\mathfrak{N}]$ to emphasise that it is the unary quantifier. We also write $\bar{Q}$ for the collection of all vectorizations of the quantifier Q. We use similar notation for the sets of numerical predicates. We use $\mathcal{L}((\mathfrak{Q})[\mathfrak{N}])$ to denote the languages expressible by the logic $(\mathfrak{Q})[\mathfrak{N}]$.

All the logics we consider are *substitution closed* in the sense of [10]. This means in particular that if a quantifier Q is definable in a logic $(\mathfrak{Q})[\mathfrak{N}]$, then extending the logic with the quantifier Q does not add to its expressive power. This is because we can replace occurrences of the quantifier Q by its definition, with a suitable substitution of the interpretation for the relation symbols. Hence, if Q is definable in $(\mathfrak{Q})[\mathfrak{N}]$, then $\mathcal{L}((\mathfrak{Q})[\mathfrak{N}]) = \mathcal{L}((\mathfrak{Q} \cup \{Q\})[\mathfrak{N}])$.

A remark is due on our notation for numerical predicates. All structures we consider are ordered, including those defining the quantifiers. Thus the order predicate is implicitly present in the collection of numerical predicates $\mathfrak{N}$ and is used (implicitly) to define the interpretations to a quantifier. We sometimes write $(\mathfrak{Q})[\varnothing]$ to indicate a logic in which this is the only use of the order that is allowed; by our choice of notation, the order symbol then does not appear explicitly in the syntax of the formulas.

2.4 Multiplication Quantifiers

The definition of multiplication quantifier has its origin in Barrington, Immerman, and Straubing [1, Section 5] where they were referred to as monoid quantifiers; the authors proved that the languages in NC^1 are exactly those expressible by first-order logic with quantifiers whose truth-value is determined via multiplication in a finite monoid. The notion was extended by Lautemann et al. [16] to include quantifiers for the word problem over more general algebras with a binary operation. Multiplication quantifiers over a finite monoid M can be understood as generalized quantifiers corresponding to languages recognized by M, and here we define them as such. We then see how this definition matches that of multiplication quantifiers *à la* Barrington et al. [1, 14].

Fix a monoid M, a set $B \subseteq M$, and a positive integer k. Let Σ_k denote the set $\{0,1\}^k$ which we think of as an alphabet of size 2^k, and fix a function $\gamma : \Sigma_k \to M$. We extend γ to strings in Σ_k^* inductively in the standard way: $\gamma(\epsilon) = 1_M$ and $\gamma(wa) = \gamma(w)\gamma(a)$. Note that γ is a monoid homomorphism. Together these define a language

$$L_\gamma^{M,B} = \{x \in \Sigma_k^* \mid \gamma(x) \in B\}.$$

We can now define a *multiplication quantifier*. In the following, $\mathcal{S}_L$ denotes the class of τ structures associated with a language L in the sense of Definition 2.

▶ **Definition 4.** *Let τ be a vocabulary including an order symbol $<$ and k unary relations. For a monoid M, a set $B \subseteq M$ and a function $\gamma : \{0,1\}^k \to M$, the* multiplication quantifier $\Gamma^{M,B}_{\gamma}$ *is the Lindström quantifier associated with the class of structures $\mathcal{S}_{L^{M,B}_{\gamma}}$.*

We also write $\Gamma^{M,B}_{d,\gamma}$ for the vectorization of this quantifier of dimension d. If B is a singleton $\{s\}$, then we may write $\Gamma^{M,s}_{d,\gamma}$ for short.

Recall that U_1 denotes the two-element monoid $\{0,1\}$ with standard multiplication. Then, it is easily seen that $\Gamma^{U_1,0}_{1,\gamma}$, where $\gamma : \{0,1\} \to U_1$ such that $\gamma(0) = 1$ and $\gamma(1) = 0$, is the standard existential quantifier. The universal quantifier can be defined similarly.

Another way of describing the semantics of the multiplication quantifier (which relates it directly to the form described in Barrington et al. [1]) is as follows. For a monoid M, a set $B \subseteq M$, a positive integer k and a function $\gamma : \{0,1\}^k \to M$, consider the formula

$$\Gamma^{M,B}_{d,\gamma}\overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))$$

where $|\overline{x}| = 2d$ and $|\overline{y}| = d$; and $I = (\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))$ defines an interpretation of dimension d from some vocabulary τ to a vocabulary with binary relation $\sqsubset$ and k unary relations. Then, for a τ-structure $\mathfrak{A}$ and assignment α, we have that

$$\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma}\overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \varphi_k(\overline{y}))[\alpha]$$

if, and only if, $\phi_<$ defines a linear order $\sqsubset$ on the d-tuples of $\mathfrak{A}$ and

$$\prod_{(\overline{a} \in ||\mathfrak{A}||^d)_{\sqsubset}} \gamma(\phi_1(\overline{a}), \ldots, \phi_k(\overline{a})) \in B.$$

Here, multiplication is in the monoid M and, since multiplication is not necessarily commutative, the order of d-tuples is specified to be the one given by $\sqsubset$, which is defined by $\varphi_<$.

As stated above, when $\mathfrak{A}$ is itself ordered and the order $\sqsubset$ is the lexicographic order on d-tuples, we simply omit the formula $\phi_<$.

▶ **Definition 5.** *For a monoid M, we define the following collections of quantifiers:*

$$\Gamma^M = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M,\ \gamma : \{0,1\}^k \to M,\ \text{and } d, k \geq 1\right\}$$
$$\Gamma^M_d = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M \text{ and } \gamma : \{0,1\}^k \to M\right\}$$
$$\Gamma^M_\gamma = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M,\ \text{and } d \geq 1\right\}$$
$$\Gamma^M_{d,\gamma} = \left\{\Gamma^{M,B}_{d,\gamma} \mid B \subseteq M\right\}$$

Finally, let Γ^{fin} be the collection of all multiplication quantifiers over finite monoids.

The study of the expressive power of multiplication quantifiers has generally been in relation to logics which restrict the application of multiplication quantifiers to interpretations with a lexicographic order. For this purpose, we introduce a piece of notation: we write lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ to mean the fragment of the logic $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ in which all applications of quantifiers from Γ^M are to interpretations with a lexicographic order. Likewise, we write fo-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ to mean the fragment of the logic $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ in which all applications of quantifiers from Γ^M are to interpretations in which the order is defined by a formula of (FO)[<].

From [1, Corollary 9.1], we know that NC^1 is characterized by (FO)[+, ×] equipped with finite multiplication quantifiers over lexicographic orders:

▶ **Theorem 6** ([1]). $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[+, \times])$.

▶ Remark 7. In fact, simply adding multiplication quantifiers for some fixed finite, non-solvable monoid to (FO)[+, ×] suffices. The definition of "non-solvable monoid" is not needed for our proofs here but, for example, the *symmetric group of degree five*, denoted S_5, is a non-solvable monoid. Therefore, we know that $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\mathrm{FO} \cup \Gamma^{S_5})[+, \times])$.

In the absence of the arithmetic predicates for addition and multiplication, the logic of multiplication quantifiers over finite monoids only allows us to define regular languages. Specifically, Barrington et al. [1, Theorem 11.1] established that the regular languages are characterized by the logic using such quantifiers with only unary interpretations.

▶ **Theorem 8** ([1]). $\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma_1^{\mathrm{fin}})[<])$.

Later, Lautemann et al. [16, Theorem 5.1] showed that allowing interpretations of higher dimension to the quantifiers does not increase the expressive power when order is the only numerical predicate.

▶ **Theorem 9** ([16]). $\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[<])$.

In Theorem 14 we show that this is true even in the presence of other numerical predicates and, therefore, Γ^{fin} can be replaced by Γ_1^{fin} even in Theorem 6. For our intended application, we need this technical result only in the case when the quantifier in Γ^{fin} is applied to interpretations where the order defined is lexicographic, and we give the proof in this special case. However, the result holds more generally for cases in which the quantifier in Γ^{fin} is applied to interpretations where the order is defined by a (FO)[<]-formula, and this may be of independent interest so we state the more general Theorem 19.

2.5 Typed Monoids

Our results build on the theory of *typed monoids* as developed in the work of Krebs and collaborators [2, 3, 14, 13, 6]. We recapitulate the main definitions in Appendix A for ease of reference. In particular, we need the notions of *division* of typed monoids (Definition 29), *syntactic typed monoid* (Definition 32), and *language recognition* with typed monoids (Definition 27). Here, we turn to discussing the relationship between the expressive power of logics with multiplication quantifiers and typed monoids. A formal association is defined through the definition below.

▶ **Definition 10.** *For a multiplication quantifier $Q = \Gamma_\gamma^{M,B}$ where $\gamma : \{0,1\}^k \to M$, we define the* typed quantifier monoid *of Q to be the syntactic typed monoid of the language $L_\gamma^{M,B}$.*

We also state the formal connection between the languages expressible in a logic with a collection of quantifiers and the corresponding class of typed monoids which recognizes the exact same language. For this we need the notion of the *ordered strong block product* of a pair of monoids. The definition is technical and can be found in [13] and is also reproduced in the linked full version of this paper. For a set of typed monoids T, we denote by $\mathrm{sbpc}_<(T)$ its closure under ordered strong block products.

From [13, Theorem 4.14], we then get the following relationship between logics and algebras:[2]

[2] The theorem in [13] is actually more general as it allows for more predicates than just order; however, for our purposes, order alone suffices.

▶ **Theorem 11.** *Let $\mathfrak{Q}$ be a collection of multiplication quantifiers and $\boldsymbol{Q}$ the set of typed quantifier monoids for quantifiers in $\mathfrak{Q}$. Then, $\mathcal{L}(\text{lex-}(\mathfrak{Q}_1)[<]) = \mathcal{L}(\text{sbpc}_<(\boldsymbol{Q}))$.*

Recall that the subscript in $\mathfrak{Q}_1$ means we are restricting the logic to only use the quantifiers in $\mathfrak{Q}$ on unary interpretations and the prefix "lex-" means that the quantifiers are only applied to interpretations with a lexicographic order.

3 Simplifying Multiplication Quantifiers

To use Theorem 11 to obtain an algebraic characterization of NC^1, we need to characterize this class in a logic with only unary quantifiers. Remark 7 gives us a characterization using first-order quantifiers and quantifiers in Γ^{S_5}. Our aim in this section is to show that we can eliminate the use of quantifiers of dimension higher than 1 in this logic. As a first step, we show that we can restrict ourselves to quantifiers $\Gamma_\delta^{S_5,B}$ for a *fixed* function δ.

▶ **Lemma 12.** *For every finite monoid M, there exists a function $\delta : \{0,1\}^{|M|} \to M$ such that for every $B \subseteq M$ and $\gamma : \{0,1\}^k \to M$, and dimension d, the quantifier $\Gamma_{d,\gamma}^{M,B}$ is definable in $(\Gamma_{d,\delta}^{M,B})[\varnothing]$.*

Proof. Recall that $\Gamma_{d,\gamma}^{M,B}$ is the class of structures $\mathfrak{A}$ in a vocabulary τ with one $2d$-ary ordering relation $<$ and k d-ary relations $R_1, \ldots, R_k$ such that $\gamma(w_\mathfrak{A}) \in B$ where $w_\mathfrak{A}$ is the $||\mathfrak{A}||^d$-length string associated with $\mathfrak{A}$ as in Def. 1.

Let $c = |M|$, fix an enumeration $\{m_1, \ldots, m_c\}$ of M. Let $\delta : \{0,1\}^c \to M$ be the function where $\delta(w) = m_i$ if w is the one-hot encoding of i and $\delta(w) = z$, for some arbitrary $z \in M$, otherwise (that is, if the number of occurrences of the symbol 1 in the string w is not exactly one).

For each $t \in M$, define the formula $\psi_t(y_1, \ldots, y_d)$ as follows:

$$\psi_t(y_1, \ldots, y_d) := \bigvee_{w \in \{0,1\}^k : \gamma(w) = t} \left(\bigwedge_{i \in [k] : w_i = 1} R_i(y_1, \ldots, y_d) \wedge \bigwedge_{i \in [k] : w_i = 0} \neg R_i(y_1, \ldots, y_d) \right).$$

It is easy to see that in a τ-structure $\mathfrak{A}$, we have $\mathfrak{A} \models \psi_t[a_1/y_1, \ldots, a_d/y_d]$ if, and only if, the element of $w_\mathfrak{A}$ indexed by $(a_1, \ldots, a_d)$ is mapped by γ to t. Thus, in particular, the formulas $\psi_{m_1}, \ldots, \psi_{m_c}$ define disjoint sets that partition the universe of $\mathfrak{A}$. We now claim that the quantifier $\Gamma_\gamma^{M,B}$ is defined by the formula:

$$\Gamma_\delta^{M,B} \overline{x_1 x_2} y_1 \ldots y_d (\overline{x_1} < \overline{x_2}, \psi_{m_1}(y_1, \ldots, y_d), \ldots, \psi_{m_c}(y_1, \ldots, y_d)).$$

To see this, let I denote the interpretation $(<, \psi_{m_1} \ldots, \psi_{m_c})$ so that $w_{I(\mathfrak{A})}$ is a string over $\{0,1\}^c$. By the fact that the sets defined by the formulas $\psi_{m_1}, \ldots, \psi_{m_c}$ partition $|\mathfrak{A}|$ it follows that each letter of $w_{I(\mathfrak{A})}$ is a vector in $\{0,1\}^c$ with exactly one 1. Indeed, the element of $w_{I(\mathfrak{A})}$ indexed by $(a_1, \ldots, a_d)$ is the one-hot encoding of i precisely if $\mathfrak{A} \models \psi_{m_i}[a_1/y_1, \ldots, a_d/y_d]$. Since δ takes the one-hot encoding of i to m_i, we have for any $a_1, \ldots, a_d \in |\mathfrak{A}|$

$$\begin{aligned} &\delta((w_{I(\mathfrak{A})})_{(a_1,\ldots,a_d)}) = m_i \\ \text{iff } &\mathfrak{A} \models \psi_i[a_1/y_1, \ldots, a_d/y_d] \\ \text{iff } &\gamma((w_\mathfrak{A})_{(a_1,\ldots,a_d)}) = m_i. \end{aligned}$$

Hence, $\delta(w_{I(\mathfrak{A})}) = \gamma(w_\mathfrak{A})$ and therefore $I(\mathfrak{A}) \in \Gamma_{d,\delta}^{M,B}$ if, and only if, $\mathfrak{A} \in \Gamma_{d,\gamma}^{M,B}$ as required. ◀

It then follows from Lemma 12 and the substitution property of quantifiers that the expressive power of $(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ is the same as that of $(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$. Indeed, any application of a quantifier in Γ^M can be replaced by an application of a quantifier in Γ^M_δ of the same dimension. We next aim to show that an application of a quantifier $\Gamma^{M,B}_{d,\delta}$ with lexicographic order, can be replaced by d nested applications of quantifiers $\Gamma^{M,B}_{1,\delta}$

▶ **Lemma 13.** *For any collection of quantifiers $\mathfrak{Q}$ and numerical predicates $\mathfrak{N}$, any formula of* lex-$(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$ *is equivalent to one of* lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$.

Proof. Again, fix an enumeration $M = \{m_1, \ldots, m_c\}$ of M and recall that $\delta : \{0,1\}^c \to M$ takes the one-hot encoding of i to m_i.

We show, by induction on d that if we have a formula

$$\Phi := \Gamma^{M,B}_{d,\delta} x_1, \ldots, x_d(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d))$$

where each formula ϕ_i is in lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$, then Φ is equivalent to a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$. The result then follows by induction on the structure of the formula.

The base case when $d = 1$ is trivially true. Assume then that we have the formula Φ for $d > 1$ and let I denote the interpretation $(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d))$ of dimension d.

We claim that Φ is equivalent to the formula

$$\Phi_1 := \Gamma^{M,B}_{1,\delta} x_1(\theta_1(x_1), \ldots, \theta_c(x_1))$$

where θ_i is the formula

$$\theta_i(x_i) := \Gamma^{M,m_i}_{d-1,\delta} x_2, \ldots, x_d(\phi_1(x_1, \ldots, x_d), \ldots, \phi_c(x_1, \ldots, x_d)).$$

Thus, Φ_1 is obtained by the application of $\Gamma^{M,B}_\delta$ to an interpretation

$$I_1 := (\theta_1(x_1), \ldots, \theta_c(x_1))$$

where each formula θ_i is obtained as the application of a quantifier $\Gamma^{M,s}_{d-1,\delta}$ to an interpretation defined by formulas of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$. Thus, by the inductive hypothesis, each θ_i is equivalent to a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$ and we are done.

It remains to show that Φ and Φ_1 are equivalent on any structure $\mathfrak{A}$. To see this, fix an assignment α of values in $|\mathfrak{A}|$ to the free variables of Φ. We need to show that $\mathfrak{A} \models \Phi[\alpha]$ if, and only if, $\mathfrak{A} \models \Phi_1[\alpha]$. Let n be the length of $\mathfrak{A}$ and assume without loss of generality that the elements of $|\mathfrak{A}|$ are $\{1, \ldots, n\}$ in that order.

Now, $w_{I(\mathfrak{A},\alpha)}$ denotes the string associated with the structure $I(\mathfrak{A}, \alpha)$ and note that this is a string of length n^d whose elements are indexed by d-tuples of elements of $\mathfrak{A}$ in lexicographic order. By definition, $\mathfrak{A} \models \Phi[\alpha]$ precisely if $\delta(w_{I(\mathfrak{A},\alpha)}) \in B$. We can also regard I as an interpretation of dimension $d-1$ obtained by treating the variable x_1 as a parameter. We write $w_{I(\mathfrak{A},\alpha[a/x_1])}$ for the string of length n^{d-1} that results from applying this interpretation with the assignment of a to the variable x_1. Since the ordering of d-tuples in $w_{I(\mathfrak{A},\alpha)}$ is lexicographic, we have

$$w_{I(\mathfrak{A},\alpha)} = w_{I(\mathfrak{A},\alpha[1/x_1])} \cdots w_{I(\mathfrak{A},\alpha[n/x_1])}$$

and thus

$$\delta(w_{I(\mathfrak{A},\alpha)}) = \delta(w_{I(\mathfrak{A},\alpha[1/x_1])}) \cdots \delta(w_{I(\mathfrak{A},\alpha[n/x_1])}).$$

Now, by definition of θ_i, we have $\mathfrak{A} \models \theta_i[\alpha[a/x_1]]$ if, and only if, $\delta(w_{I(\mathfrak{A},\alpha[a/x_1])}) = m_i$. Thus, for each $a \in |\mathfrak{A}|$, there is exactly one i such that $\mathfrak{A} \models \theta_i[\alpha[a/x_1]]$. Thus, $w_{I_1(\mathfrak{A},\alpha)}$ is the string of length n whose ath element is the one-hot encoding of i exactly when $\delta(w_{I(\mathfrak{A},\alpha[a/x_1])}) = m_i$. In other words, $\delta((w_{I_1(\mathfrak{A},\alpha)})_a) = \delta(w_{I(\mathfrak{A},\alpha[a/x_1])})$. Thus,

$$
\begin{aligned}
&\mathfrak{A} \models \Phi[\alpha] \\
&\text{iff } \delta(w_{I(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \delta(w_{I(\mathfrak{A},\alpha[1/x_1])}) \cdots \delta(w_{I(\mathfrak{A},\alpha[n/x_1])}) \in B \\
&\text{iff } \delta((w_{I_1(\mathfrak{A},\alpha)})_1) \cdots \delta((w_{I_1(\mathfrak{A},\alpha)})_n) \in B \\
&\text{iff } \delta(w_{I_1(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \mathfrak{A} \models \Phi_1[\alpha].
\end{aligned}
$$

◀

Now we are ready to state the main theorem of this section.

▶ **Theorem 14.** *For every finite monoid M, there exists a function $\delta : \{0,1\}^{|M|} \to M$ such that for any collection of quantifiers $\mathfrak{Q}$ and any set $\mathfrak{N}$ of numerical predicates, every formula of* lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$ *is equivalent to a formula of* lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$.

Proof. Let ϕ be any formula of lex-$(\mathfrak{Q} \cup \Gamma^M)[\mathfrak{N}]$. By Lemma 12 we can replace all occurrences of quantifiers in $\Gamma^{M,B}_\gamma$ by their definitions using quantifiers in $\Gamma^{M,B}_\delta$ to get a formula ϕ' of lex-$(\mathfrak{Q} \cup \Gamma^M_\delta)[\mathfrak{N}]$ equivalent to ϕ. Finally, by Lemma 13, there is a formula of lex-$(\mathfrak{Q} \cup \Gamma^M_{1,\delta})[\mathfrak{N}]$ equivalent to ϕ'. ◀

Note that for a finite monoid M, while Γ^M and Γ^M_1 are infinite sets, $\Gamma^M_{1,\delta}$ is a finite set. Therefore, this gives us a logic characterizing NC^1 which not only uses unary quantifiers but also only has a finite number of quantifiers:

▶ **Corollary 15.** *There exists a $\delta : \{0,1\}^k \to S_5$ such that* $\mathrm{NC}^1 = \mathcal{L}(\text{lex-}(\mathrm{FO} \cup \Gamma^{S_5}_{1,\delta})[+,\times])$.

This simplifies our construction of an algebra capturing NC^1.

Another consequence of Theorem 14 is Theorem 9 ([16, Theorem 5.1]). Thus, we get a proof of Theorem 9 which is purely logical, unlike the original proof which relies on the use of finite automata. Furthermore, we also resolve a question left open in Lautemann et al. [16]:

▶ **Corollary 16.** $\mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}})[+,\times]) = \mathcal{L}(\text{lex-}(\Gamma^{\mathrm{fin}}_1)[+,\times])$

4 The Algebraic Characterization

Now that we have an extension of first-order logic capturing NC^1 using only unary quantifiers, we are able to apply Theorem 11 to construct an algebra for it. We just need to obtain an equivalent logic using only the numerical predicate $<$ without introducing quantifiers of higher dimension.

To do this, we follow the construction of an algebra for TC^0. The *majority* quantifier Maj is the collection of strings over the alphabet $\{0,1\}$ in which at least half of the symbols are 1. The *square* quantifier Sq is the collection of strings over the alphabet $\{0,1\}$ in which the number of 1s is a positive square number (i.e., an element of $\mathbb{S}$). In the logics we define below, we always use these quantifiers only with unary interpretations.

The following lemma displays some known results about the expressiveness of these quantifiers:

▶ **Lemma 17.**

(i) Maj *is definable in* lex-$(\Gamma^{\text{fin}})[+,\times]$. *(cf. [1])*

(ii) *The quantifiers in* FO *are definable in* $(\text{Maj})[<]$. *([15, Theorem 3.2])*

(iii) *The numerical predicate* $+$ *is definable in* $(\text{Maj})[<]$. *([15, Theorem 4.1])*

(iv) *The numerical predicate* $\times$ *is definable in* $(\{\text{Maj},\text{Sq}\})[<]$ *and* Sq *is definable in* $(\text{Maj})[<,+,\times]$. *(cf. [18, Theorem 2.3.f] and [14, Section 2.3])*

Bringing everything together, we get the following algebraic characterization of NC^1:

▶ **Theorem 18.**

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\}),(S_5,\wp(S_5),S_5)\})).$$

Proof. Let $\delta : \{0,1\}^c \to S_5$ be as defined in Lemma 12. It is easy to see that the typed quantifier monoid for Maj is $(\mathbb{Z},\mathbb{Z}^+,\pm 1)$, for Sq is $(\mathbb{N},\mathbb{S},\{0,1\})$, and for $\Gamma_{1,\delta}^{S_5,s}$ is $(S_5,\{s\},S_5)$ for any $s \in S_5$.

By Corollary 15, we have that $\text{NC}^1 = \mathcal{L}(\text{lex-}(\text{FO} \cup \Gamma_{1,\delta}^{S_5})[+,\times])$. Lemma 17 (ii)–(iv) allow us to define FO, $+$, and $\times$ in $(\{\text{Maj},\text{Sq}\})[<]$ and (i) and (iv) allow us to define Maj and Sq in lex-$(\text{FO} \cup \Gamma_{1,\delta}^{S_5})[+,\times]$. Therefore, $\text{NC}^1 = \mathcal{L}(\text{lex-}(\Gamma_{1,\delta}^{S_5} \cup \{\text{Maj},\text{Sq}\})[<])$. Moreover, we may restrict this to just quantifiers $\Gamma^{S_5,A}$ with A a singleton set, as for any $A \subseteq S_5$, the quantifier $\Gamma_{1,\delta}^{S_5,A}$ may be easily defined using Boolean combinations of quantifiers $\Gamma_{1,\delta}^{S_5,s}$ since S_5 is finite. Theorem 11 then gives us the algebraic characterization of

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\})\} \cup \{(S_5,s,S_5) \mid \{s\} \in \wp(S_5)\})).$$

Because $(S_5,s,S_5) \prec (S_5,\wp(S_5),S_5)$ for all $\{s\} \in \wp(S_5)$, we lose no expressive power by replacing all elements of the form (S_5,s,S_5) with $(S_5,\wp(S_5),S_5)$. We neither gain expressive power because $\mathcal{L}((S_5,\wp(S_5),S_5)) \subseteq \text{REG} \subseteq \text{NC}^1$. Therefore, we have our final characterization:

$$\text{NC}^1 = \mathcal{L}(\text{sbpc}_<(\{(\mathbb{Z},\mathbb{Z}^+,\pm 1),(\mathbb{N},\mathbb{S},\{0,1\}),(S_5,\wp(S_5),S_5)\})).$$ ◀

5 For-Programs and Non-Lexicographic Interpretations

The proof of Lemma 13 relies crucially on the fact that we only allow lexicographic ordering of tuples in our interpretations. This is sufficient for the algebraic characterization in Section 4. However, in this section, we prove a more general version: we prove that even when interpretations are allowed to use any first-order definable ordering of tuples, we can replace multiplication quantifiers of higher arity by unary quantifiers. To do so, we rely on the work of Bojańczyk et al. [5], which characterizes first-order definable orders in terms of *d-enumerators* and *for-programs* which we review below. These are combined with techniques introduced in the proof of Theorem 14 to prove the following:

▶ **Theorem 19.** *Let* $\mathfrak{Q}$ *be any collection of quantifiers and* $\mathfrak{N}$ *any collection of numerical predicates. Then,* $\mathcal{L}(\text{fo-}(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]) = \mathcal{L}(\text{lex-}(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}])$.

As we noted in Section 2, we always assume that the collection $\mathfrak{N}$ of numerical predicates contains the order relation $<$, except in the case where it is empty. Thus, the statement of the theorem notes that in translating our formulas to use unary quantifiers, we may need to introduce the order symbol if $\mathfrak{N} = \emptyset$. From the theorem, we then get the immediate corollaries.

▶ **Corollary 20.** $\mathrm{NC}^1 = \mathcal{L}(\text{fo-}(\Gamma^{\text{fin}})[+,\times])$.

▶ **Corollary 21.** $\textsc{Reg} = \mathcal{L}(\text{fo-}(\Gamma^{\text{fin}})[<])$.

Before going into the proof, we first introduce for-programs. As the definition of d-enumerators and for-programs in general are relatively intuitive, we keep the definition brief and a more detailed treatment may be found in [5, 4]. A *first-order d-enumerator* is a for-program which takes a structure $\mathfrak{A} = (A, <, \tau^{\mathfrak{A}})$ over vocabulary τ, as input and outputs an enumeration of all the d-tuples of A. The program consists of a nesting of *for-loops* with a body:

```
for y_1 in p_1
    for y_2 in p_2
        ...
            for y_{d'} in p_{d'}
                body
```
(1)

Here the ith loop iterates the variable y_i over the domain A and p_i determines whether the iteration is in increasing order ($p_i =$ `first..last`) or decreasing order ($p_i =$ `last..first`), according to the order $<$.

The body consists of a sequence of *if-statements*

```
if θ_1(y_1,...,y_{d'}) then
    output (y_{i_1^1},...,y_{i_d^1})
if θ_2(y_1,...,y_{d'}) then
    output (y_{i_1^2},...,y_{i_d^2})
...
if θ_l(y_1,...,y_{d'}) then
    output (y_{i_1^l},...,y_{i_d^l})
```
(2)

Here $\theta_1, \ldots, \theta_l$ are $(\text{FO})[<]$-formulas over the vocabulary τ, specifying mutually exlusive conditions, so that for each assignment of values to the variables $y_1, \ldots, y_{d'}$ at most one of them is satisfied. Note that which formula is satisfied may depend on the assignment α of values to free variables other than $y_1, \ldots, y_{d'}$. We only consider programs which produce, over the course of the iteration, all tuples in A^d, with each tuple being output exactly once. For a string $\mathfrak{A}$ and assignment α, we write $P(\mathfrak{A}, \alpha)$ to mean the output of the program P when run on the string $\mathfrak{A}$ with assignment α.

From Bojańczyk et al. [5, Theorem 12], we know the following fact:

▶ **Lemma 22** ([5])**.** *For every* $(\text{FO})[<]$ *definable linear order on d-tuples, defined by a formula $\phi_<$ over a vocabulary τ of unary predicates, there exists a first-order d-enumerator P such that for every τ-structure $\mathfrak{A} = (A, <, \tau^{\mathfrak{A}})$, P on input $\mathfrak{A}$ enumerates the elements of A^d in the order defined by $\phi_<$.*

We now use this to prove Theorem 19. We first use Lemma 22 to transform a formula of $(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]$ into one in which all applications of quantifiers in Γ^{fin} use interpretations using a *generalized lexicographic order*, and then use the techniques introduced in Section 3 to transform this to a formula using only unary quantifiers.

▶ **Definition 23.** *Given a set A with a linear order $<$ on its elements and $d \in \mathbb{Z}$, a linear order $\prec$ on A^d is called a* generalized lexicographic order *if there is a sequence* $\text{dir} \in \{l, r\}^d$ *such that $\vec{a} \prec \vec{b}$ if, and only if, for the least i for which $a_i \neq b_i$, we have $a_i < b_i$ if $\text{dir}_i = l$ and $a_i > b_i$ if $\text{dir}_i = r$.*

Note that the standard lexicographic order is a generalized lexicographic order with $\text{dir} = l^d$.

Proof of Theorem 19. Let $\mathfrak{Q}$ be any collection of quantifiers and $\mathfrak{N}$ any collection of numerical predicates. Let $M = \{m_1, \ldots, m_c\}$ be a finite monoid, $B \subseteq M$, $\gamma : \{0,1\}^k \to M$, and $\phi_<$ a (FO)[<]-formula over a vocabulary τ which defines a linear ordering on d-tuples. Let $\phi_1, \ldots, \phi_k$ be fo-$(\mathfrak{Q} \cup \Gamma^{\text{fin}})[\mathfrak{N}]$-formulas over τ giving an interpretation $I = (\phi_<, \phi_1, \ldots, \phi_k)$ of dimension d. We want to show that for

$$\Phi := \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi_1(\overline{y}), \ldots, \phi_k(\overline{y}))[\alpha],$$

there is an equivalent formula in lex-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$. We first prove by induction on the nesting of multiplication quantifiers that Φ is equivalent to a formula in which all applications of quantifiers in Γ^{fin} are to interpretations in which the order defined is a generalized lexicographic order. Note that formulas (e.g., $\phi_<$) defining the order in interpretations are in (FO)[<] and, thus, by assumption do not have nested multiplication quantifiers.

For the base case, say we have a formula which uses no multiplication quantifiers; then, we are done as this is a formula of lex-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$. For the inductive step, assume that $\phi_1, \ldots, \phi_k$ are formulas of fo-$(\mathfrak{Q} \cup \Gamma_1^{\text{fin}})[\mathfrak{N} \cup \{<\}]$ in which all orders are generalized lexicographical orders.

By Lemma 22, we have a first-order d-enumerator P which enumerates the elements of A^d in the order defined by $\phi_<$ on any τ-structure $\mathfrak{A}$. P has the form specified in (1) and (2). Recall that in this for-program we have l if-statements. Now, let Φ' be the formula

$$\begin{aligned}\Phi' := \Gamma^{M,B}_{d',\delta} \overline{x} y_1 \ldots y_{d'} (\xi_<(\overline{x}), &\xi_1(y_1, \ldots, y_{d'}), \ldots, \xi_c(y_1, \ldots, y_{d'}),\\ &\xi_{c+1}(y_1, \ldots, y_{d'}), \ldots, \xi_{2c}(y_1, \ldots, y_{d'}),\\ &\ldots,\\ &\xi_{(l-1)c+1}(y_1, \ldots, y_{d'}), \ldots, \xi_{lc}(y_1, \ldots, y_{d'}))\end{aligned}$$

where $\delta : \{0,1\}^{lc} \to M$ is such that for a word $w = w_1 \ldots w_{lc} \in \{0,1\}^{lc}$, $\delta(w) = m_i$ if there exists a j where $0 \leq j < l$ such that the substring $w_{jc+1} \ldots w_{(j+1)c}$ is equal to the one-hot encoding of i and all other characters in w are 0. In other words, partitioning w into l consecutive c-length substrings, we map w to m_i if one of these substrings is the one-hot encoding of i and all others are simply strings of 0s. All other elements of the domain are mapped to the identity of M. Moreover, for $0 \leq j < l$ and $1 \leq j' \leq c$, $\xi_{jc+j'}(y_1, \ldots, y_{d'})$ is the formula $\theta_j(y_1, \ldots, y_{d'}) \wedge \psi_{m_{j'}}(y_{i_1^j}, \ldots, y_{i_d^j})$ where $i_1^j, \ldots, i_d^j$ are defined as they are in the for-program P, and $\psi_{m_{j'}}$ is as defined in Lemma 12; therefore, $\xi_{jc+j'}$ is satisfied if both $\theta_j(y_1, \ldots, y_{d'})$ is and the element of M indexed by $(y_{i_1^j}, \ldots, y_{i_d^j})$ equals $m_{j'}$. Lastly, $\xi_<$ is the generalized lexicographic order given by the vector $\text{dir} \in \{l, r\}^{d'}$ with $\text{dir}_i = l$ if $p_i = \texttt{first..last}$ and $\text{dir}_i = r$ if $p_{d'} = \texttt{last..first}$. We now prove that $\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha]$ iff $\mathfrak{A} \models \Phi'[\alpha]$.

Let 1_M denote the identity of M and $f_{\mathfrak{A},\alpha} : A^d \cup \{\epsilon\} \to M$ be the function defined by

$$f_{\mathfrak{A},\alpha}(a_1, \ldots, a_d) = \gamma({\phi'_1}^{\mathfrak{A},\alpha}[a_1, \ldots, a_d], \ldots, {\phi'_k}^{\mathfrak{A},\alpha}[a_1, \ldots, a_d])$$

for $(a_1, \ldots, a_d) \in A^d$ and $f_{\mathfrak{A},\alpha}(\epsilon) = 1_M$. and let $g_{\mathfrak{A},\alpha} : A^{d'} \to A^d \cup \{\epsilon\}$ be the function defined by the body of the for-program P where if no output is produced, $g_{\mathfrak{A},\alpha}(a_1, \ldots, a_{d'}) = 0^d$; thus,

$$g_{\mathfrak{A},\alpha}(a_1, \ldots, a_{d'}) = \begin{cases} (a_{i_1^j}, \ldots, a_{i_d^j}) & \text{if } \exists j \text{ such that } \mathfrak{A} \models \theta_j(a_1, \ldots, a_{d'}) \\ \epsilon & \text{o.w.} \end{cases}$$

Let $f^*_{\mathfrak{A},\alpha} : (A^d)^* \to M$ and $g^*_{\mathfrak{A},\alpha} : (A^{d'})^* \to (A^d)^*$ be the monoid homomorphisms induced by the functions $f_{\mathfrak{A},\alpha}$ and $g_{\mathfrak{A},\alpha}$, respectively, in the natural way.

Let $\mathfrak{A}$ be an arbitrary τ-structure and α an arbitrary variable assignment with assignments for at least all free variables in each ϕ'_i with the exception of $\overline{y}_i$. By definition of multiplication quantifiers, we get that $\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha]$ iff $\gamma(w_{I(\mathfrak{A},\alpha)}) \in B$.

By construction of $f_{\mathfrak{A},\alpha}$ and because P provides the order by which $\Gamma^{M,B}_{d,\gamma}$ is evaluated, it is easy to see that $\gamma(w_{I(\mathfrak{A},\alpha)}) \in B$ iff $f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B$.

Let $t_1, \ldots, t_{|A|^{d'}}$ denote the ordering of d'-tuples of A as defined by $\xi_<$. Because the output of P is an enumeration of A^d in the ordering defined by $\phi_<$, and $g_{\mathfrak{A},\alpha}$ outputs the empty string ϵ during an iteration of its for-loops only when P outputs nothing at that iteration, it follows that

$$f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) = f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha))$$

and, thus, $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B$ iff $f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B$.

Let $I' = (\xi_<, \xi_1, \ldots, \xi_{lc})$. We now want to show that

$$\delta(w_{I'(\mathfrak{A},\alpha)}) \in B \text{ iff } f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B.$$

To do so, we prove that $\delta((w_{I'(\mathfrak{A},\alpha)})_a) = f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_a))$ for every $0 \leq a \leq |A|^{d'}$. Let a be arbitrary. Then, $u = (w_{I(\mathfrak{A},\alpha)})_a$ is either 0^{lc} or a one-hot encoding of some $b \in [lc]$. If $u = 0^{lc}$, then $\delta(u) = 1_M$ and no θ_j is satisfied. Thus, $g^*_{\mathfrak{A},\alpha}(t_a) = \epsilon$ and $f^*_{\mathfrak{A},\alpha}(\epsilon) = 1_M$ by construction. If u is a one-hot encoding of some $b \in [lc]$, then some θ_j is satisfied and,

$$\delta(u) = m = \gamma({\phi'_1}^{\mathfrak{A},\alpha}[a_{i_1^j}, \ldots, a_{i_d^j}], \ldots, {\phi'_k}^{\mathfrak{A},\alpha}[a_{i_1^j}, \ldots, a_{i_d^j}]) = f_{\mathfrak{A},\alpha}(a_{i_1^j}, \ldots, a_{i_d^j})$$

by definition of δ and the construction of each ξ_i. By the construction of $g_{\mathfrak{A},\alpha}$ and $f_{\mathfrak{A},\alpha}$, we also get immediately that $g^*_{\mathfrak{A},\alpha}(t_a) = (a_{i_1^j}, \ldots, a_{i_d^j})$ and, thus, $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_a)) = m$. Thus, $\delta(w_{I'(\mathfrak{A},\alpha)}) \in B$ iff $f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{|A|^{d'}})) \in B$.

Then, by definition, we have that $\mathfrak{A} \models \Phi'[\alpha]$ iff $\delta(w_{I'(\mathfrak{A},\alpha)}) \in B$.

Therefore, all together, we have that

$$\begin{aligned}
&\mathfrak{A} \models \Gamma^{M,B}_{d,\gamma} \overline{xy}(\phi_<(\overline{x}), \phi'_1(\overline{y}), \ldots, \phi'_k(\overline{y}))[\alpha] \\
&\text{iff } \gamma(w_{I(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } f^*_{\mathfrak{A},\alpha}(P(\mathfrak{A},\alpha)) \in B \\
&\text{iff } f^*_{\mathfrak{A},\alpha}(g^*_{\mathfrak{A},\alpha}(t_1 \ldots t_{A^{d'}})) \in B \\
&\text{iff } \delta(w_{I'(\mathfrak{A},\alpha)}) \in B \\
&\text{iff } \mathfrak{A} \models \Phi''[\alpha].
\end{aligned}$$

This concludes the inductive step. Therefore, for any formula Φ, we have an equivalent formula Φ' which uses only uses multiplication quantifiers using a generalized lexicographic order.

To see that we can construct a formula equivalent to Φ' using only unary quantifiers, we use the technique used in the proof of Lemma 13 with a minor modification to account for generalized lexicographic order. In the proof of Lemma 13, a quantifier $\Gamma^{M,B}_{d,\gamma}$ of dimension d is replaced by a sequence of d nested unary quantifiers over the same monoid M. To account for the order $\xi_<$, which defines a generalized lexicographic order with $\text{dir} \in \{l, r, \}^d$, we define the ith quantifier in the sequence as before if $\text{dir}_i = l$ and replace the monoid $(M, \cdot)$ with $(M, \cdot^R)$ where $m \cdot^R m' := m' \cdot m$ for all $m, m' \in M$ if $\text{dir}_i = r$. The proof follows exactly along the lines of the proof of Lemma 13.

All together, we have an equivalent formula to Φ', and, thus, Φ, which only uses unary quantifiers, completing the proof. ◀

6 Conclusion

In this work, we constructed a class of typed monoids exactly recognizing NC^1. To do so, we proved results regarding the expressive power of logics with quantifiers defined over finite monoids. Specifically, we established that the expressive power is not changed by restricting the dimension of the interpretations on which the quantifiers act, regardless of which numerical predicates are available.

Therefore, we were able to provide a logic characterizing NC^1 which only uses unary quantifiers and use this logic to construct an algebraic characterization of NC^1. This result marks the second circuit complexity class to be characterized in a such a way, with the first being TC^0 [14], and provides a decomposition theorem in the style of Krohn-Rhodes for NC^1.

An interesting future direction would be to construct similar algebraic characterization of other complexity classes beyond NC^1. It seems this would require the development of new algebraic tools. In particular, the block product is a key tool used to characterize first-order quantification and more generally quantification over interpretations of dimension one. To extend the work to other complexity classes, it would be worthwhile investigating other product constructions that might similarly relate to quantification on interpretations of higher dimension as well as higher-order quantifiers.

Moreover, we extended the work of Barrington et al. [1] and Lautemann et al. [16] by studying the expressive power of multiplication quantifiers when applied to interpretations using other than the standard lexicographic order. We showed that over strings the expressive power of logics equipped with finite multiplication quantifiers is not changed by loosening the restriction to permitting any first-order definable linear order. A natural next step would be to investigate the expressive power when other linear orders are permitted. For example, by extending the Domination Lemma of Bojańczyk et al. [5] to monadic second-order logic (MSO[<]), one would be able to show that permitting the application of multiplication quantifiers to interpretations with an MSO[<]-definable order does not change the expressive power. This would result in the interesting consequence that

$$\textsc{Reg} = \mathcal{L}(\text{lex-}(\Gamma_1^{\text{fin}})[<]) = \mathcal{L}((\Gamma^{\text{fin}})[<]).$$

References

1 David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

2 Christoph Behle, Andreas Krebs, and Mark Mercer. Linear circuits, two-variable logic and weakly blocked monoids. In *International Symposium on Mathematical Foundations of Computer Science*, pages 147–158. Springer, 2007. doi:10.1007/978-3-540-74456-6_15.

3 Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid. Typed monoids–An Eilenberg-like theorem for non regular languages. In *Algebraic Informatics: 4th International Conference, CAI 2011, Linz, Austria, June 21-24, 2011. Proceedings 4*, pages 97–114. Springer, 2011. doi:10.1007/978-3-642-21493-6_6.

4 Mikolaj Bojańczyk. Transducers of polynomial growth. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–27, 2022.

5 Mikolaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, page 106. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

6 Antonio Cano, Jesus Cantero, and Ana Martínez-Pastor. A positive extension of Eilenberg's variety theorem for non-regular languages. *Applicable Algebra in Engineering, Communication and Computing*, 32(5):553–573, 2021. doi:10.1007/S00200-020-00414-2.

7 Stephen A Cook. Characterizations of Pushdown Machines in Terms of Time-Bounded Computers. *Journal of the ACM (JACM)*, 18(1):4–18, 1971. doi:10.1145/321623.321625.

8 Anuj Dawar and Aidan T Evans. Characterizing NC1 with typed monoids. *arXiv preprint arXiv:2508.11019*, 2025. doi:10.48550/arXiv.2508.11019.

9 H-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

10 Heinz-Dieter Ebbinghaus. Extended logics: The general framework. In Jon Barwise and Solomon Feferman, editors, *Model-Theoretic Logics*, pages 25–76. Springer-Verlag, New York, 1985.

11 Samuel Eilenberg. *Automata, Languages, and Machines (Vol. B)*. Academic Press, 1976.

12 Fred C Hennie. One-tape, off-line turing machine computations. *Information and Control*, 8(6):553–578, 1965. doi:10.1016/S0019-9958(65)90399-2.

13 Andreas Krebs. *Typed Semigroups, Majority Logic, and Threshold Circuits*. PhD thesis, Tübingen, Univ., Diss., 2008, 2008.

14 Andreas Krebs, Klaus-Jörn Lange, and Stephanie Reifferscheid. Characterizing TC0 in terms of infinite groups. *Theory of Computing Systems*, 40(4):303–325, 2007. doi:10.1007/S00224-006-1310-2.

15 Klaus-Jörn Lange. Some results on majority quantifiers over words. In *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, pages 123–129. IEEE, 2004. doi:10.1109/CCC.2004.1313817.

16 Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *Journal of Computer and System Sciences*, 62(4):629–652, 2001. doi:10.1006/JCSS.2000.1742.

17 John L Rhodes. *Applications of Automata Theory and Algebra: Via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*. World Scientific, 2010.

18 Nicole Schweikardt. *On the Expressive Power of First-order Logic with Built in Predicates*. Logos-Verlag, 2002.

19 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

20 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Science & Business Media, 1999.

A Typed Monoids

In this section, we review the definitions and results from [3, 13] on typed monoids, their relationship to languages and corresponding characterizations of complexity classes.

A typed monoid is a monoid equipped with a collection of *types*, which form a Boolean algebra, and a set of *units*. We only deal with concrete Boolean algebras, given as collections of subsets of a fixed universe.

▶ **Definition 24** (Boolean Algebra). *A* Boolean algebra *over a set S is a set $B \subseteq \wp(S)$ such that $\varnothing, S \in B$ and B is closed under union, intersection, and complementation. If B is finite, we call it a* finite *Boolean algebra.*

We call $\varnothing$ and S the trivial elements *(or in some contexts, the* trivial types*) of B.*

A homomorphism between Boolean algebras is defined as standard. That is, if B_1 and B_2 are Boolean algebras over sets S and T, respectively, then we call $h : B_1 \to B_2$ a *homomorphism* if $h(\varnothing) = \varnothing$, $h(S) = T$, and for all $s_1, s_2 \in B_1$, $h(s_1 \cap s_2) = h(s_1) \cap h(s_2)$, $h(s_1 \cup s_2) = h(s_1) \cup h(s_2)$, and $h(s^C) = (h(s))^C$. Now we are ready to define typed monoids.

▶ **Definition 25** (Typed Monoid). *Let M be a monoid, G a Boolean algebra over M, and E a finite subset of M. We call the tuple $T = (M, G, E)$ a* typed monoid over M *and the elements of G* types *and the elements of E* units*. We call M the* base monoid *of T. If M is a group, then we may also call T a* typed group*.*

When $G = \{\varnothing, A, M - A, M\}$ for some $A \subseteq M$, we abbreviate T as (M, A, E), i.e., the Boolean algebra is signified by an element, or elements, which generates it – in this case, A.

We say that a typed monoid (M, G, E) is finite if M is.

We also need a notion of morphism between typed monoids.

▶ **Definition 26.** *A* typed monoid homomorphism h *from* (S, G, E) *to* (T, H, F) *is a triple* (h_1, h_2, h_3) *where* $h_1 : S \to T$ *is a monoid homomorphism,* $h_2 : G \to H$ *is a homomorphism of Boolean algebras, and* $h_3 : E \to F$ *is a mapping of sets such that the following conditions hold:*

(i) *For all* $A \in G$, $h_1(A) = h_2(A) \cap h_1(S)$.

(ii) *For all* $e \in E$, $h_1(e) = h_3(e)$.

Note that h_3 is redundant in the definition as it is completely determined by h_1. We retain it as part of the definition for consistency with [3, 13].

To motivate the definitions, recall that a language $L \subseteq \Sigma^*$ is recognized by a monoid M if there is a homomorphism $h : \Sigma^* \to M$ and a set $B \subseteq M$ such that $L = h^{-1}(B)$. When the monoid M is infinite, the languages recognized form a rather rich collection and we aim to restrict this in two ways. First, B cannot be an arbitrary set but must be an element of the algebra of types. Secondly, the homomorphism h must map the letters in Σ to units of the typed monoid. Formally, we have the following definition.

▶ **Definition 27.** *A typed monoid* $T = (M, G, E)$ recognizes *a language* $L \subseteq \Sigma^*$ *if there exists a typed monoid homomorphism from* (Σ^*, L, Σ) *to* T. *We let* $\mathcal{L}(T)$ *denote the set of languages recognized by* T.

When the base monoid of a typed monoid is finite, we recover the classical definition of a recognition. Hence, the languages recognized by finite typed monoids are necessarily regular.

▶ **Proposition 28.** *If* T *is a finite typed monoid, then* $\mathcal{L}(T) \subseteq$ Reg.

We can now state the definitions of the key relationships between typed monoids.

▶ **Definition 29.** *Let* (S, G, E) *and* (T, H, F) *be typed monoids.*

- *A typed monoid homomorphism* $h = (h_1, h_2, h_3) : (S, G, E) \to (T, H, F)$ *is* injective *(*surjective*, or* bijective*) if all of* h_1, h_2, *and* h_3 *are.*
- (S, G, E) *is a* typed submonoid *(or, simply, "submonoid" when context is obvious) of* (T, H, F), *denoted* $(S, G, E) \leq (T, H, F)$, *if there exists an injective typed monoid homomorphism* $h : (S, G, E) \to (T, H, F)$.
- (S, G, E) *divides* (T, H, F), *denoted* $(S, G, E) \preceq (T, H, F)$, *if there exists a surjective typed monoid homomorphism from a submonoid of* (T, H, F) *to* (S, G, E).

These have the expected properties.

▶ **Proposition 30** ([3])**.** *Let* T_1, T_2, *and* T_3 *be typed monoids.*

- *Typed monoid homomorphisms are closed under composition.*
- *Division is transitive: if* $T_1 \preceq T_2$ *and* $T_2 \preceq T_3$, *then* $T_1 \preceq T_3$.
- *If* $T_1 \preceq T_2$, *then* $\mathcal{L}(T_1) \subseteq \mathcal{L}(T_2)$.

We can formulate the notion of the *syntactic typed monoid* of a language L as an extension of the syntactic monoid of L with a minimal collection of types and units necessary.

▶ **Definition 31.** *Let $T = (M, G, E)$ be a typed monoid. A congruence $\sim$ over M is a* typed congruence over T *if for every $A \in G$ and $s_1, s_2 \in M$, if $s_1 \sim s_2$ and $s_1 \in A$, then $s_2 \in A$.*

For a typed congruence $\sim$ over T, let

$$A/{\sim} = \{[x]_\sim \mid x \in A\} \text{ where } A \subseteq M$$
$$G/{\sim} = \{A/{\sim} \mid A \in G\}$$
$$E/{\sim} = \{[x]_\sim \mid x \in E\}.$$

Then, $T/{\sim} := (M/{\sim}, G/{\sim}, E/{\sim})$ is the typed quotient monoid of T by $\sim$.

Let $\sim_T$ denote the typed congruence on T such that for $s_1, s_2 \in S$, $s_1 \sim_T s_2$ if, and only if, for all $x, y \in S$ and $A \in G$, $xs_1y \in A$ if, and only if, $xs_2y \in A$. We then refer to the quotient monoid $T/{\sim_T}$ as the minimal reduced monoid of T.

Recall that $\sim_L$ is the syntactic congruence of L, defined in Section 2.2.

▶ **Definition 32.** *For a language $L \subseteq \Sigma^*$, the* syntactic typed monoid of L, *denoted* $\mathrm{syn}(L)$, *is the typed monoid $(\Sigma^*, L, \Sigma)/{\sim_L}$.*

We also get the *canonical typed monoid homomorphism*, $\eta_L : (\Sigma^*, L, \Sigma) \to \mathrm{syn}(L)$ induced by the syntactic homomorphism of L.

It also turns out that we can give a purely structural characterization of those typed monoids that are syntactic monoids.

▶ **Proposition 33** ([13])**.** *A typed monoid is the syntactic monoid of a language if, and only if, it is reduced, generated by its units, and has four or two types.*

In case it has just two types, then it only recognizes the empty language or the language of all strings.

Fault-Tolerant Approximate Distance Oracles with a Source Set

Dipan Dey
Tata Institute of Fundamental Research, Mumbai, India

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India

Abstract

Our input is an undirected weighted graph $G = (V, E)$ on n vertices along with a source set $S \subseteq V$. The problem is to preprocess G and build a compact data structure such that upon query $Qu(s, v, f)$ where $(s, v) \in S \times V$ and f is any faulty edge, we can quickly find a good estimate (i.e., within a small multiplicative stretch) of the s-v distance in $G - f$. We use a fault-tolerant ST-distance oracle from the work of Bilò et al. (STACS 2018) to construct an $S \times V$ approximate distance oracle or *sourcewise* approximate distance oracle of size $\widetilde{O}(|S|n + n^{3/2})$ with multiplicative stretch at most 5. We construct another fault-tolerant sourcewise approximate distance oracle of size $\widetilde{O}(|S|n + n^{4/3})$ with multiplicative stretch at most 13. Both the oracles have $O(1)$ query answering time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Weighted graphs, approximate distances, fault-tolerant data structures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.27

Funding Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

1 Introduction

The problem of computing distances between all pairs of vertices in a given graph $G = (V, E)$ with positive edge weights is a fundamental problem in graph algorithms. The problem here is to preprocess G and build a compact data structure (called a *distance oracle*) that can quickly answer distance queries for any pair of vertices. As is often the case with real-world networks like routing networks or road networks, links may fail or roads may be temporarily blocked. Thus we have to allow for the case of faulty edges. Since several links are unlikely to fail simultaneously, we consider the case of a single edge failure.

Instead of recomputing distances from scratch for all pairs of vertices after an edge has failed, the problem is to build a resilient data structure that can answer distance queries between vertices after a single edge failure. Furthermore, we assume there is a specific set $S \subseteq V$ of sources, e.g., S is a set of starting locations on a road network or S is a set of source nodes in a routing network. Thus we are interested in distances only for pairs $(s, v) \in S \times V$. Suppose $|S| \ll n$, say $O(n^\epsilon)$ for some $\epsilon \in (0, 1)$. Then it feels wasteful to build a fault-tolerant distance oracle that maintains distances for all pairs of vertices. Another application is Vickrey pricing [21], with the objective of determining, for every $(s, v) \in S \times V$ where S is a given subset of V and every edge f, how much the distance from s to v increases if f were to fail.

Thus the distance oracle has to process queries of the form (s, v, f) where s is the source, v is the destination, and f is the failed edge. Upon query $Qu(s, v, f)$, the oracle has to return the s to v distance in $G - f$, where $G - f$ is the graph obtained by removing edge f from G. So our problem is the following (as described below).


© Dipan Dey and Telikepalli Kavitha;
licensed under Creative Commons License CC-BY 4.0

- Preprocess G and build a compact data structure that can quickly answer distance queries for any pair in $S \times V$ when an edge fails. Hence our distance oracle has to answer queries $Qu(s, v, f)$ where $s \in S, v \in V$, and f is the failed edge.

This data structure is called a *single edge fault-tolerant sourcewise distance oracle*. As discussed below, for undirected unweighted graphs, a single edge fault-tolerant sourcewise exact distance oracle of size $\tilde{O}(n^{3/2}\sqrt{|S|})$ with $\tilde{O}(1)$ query time is known [20]. Our goal is to design more compact distance oracles (for sublinear sets S) in weighted graphs. Furthermore, for the sake of space efficiency, we are ready to relax *exactness*. Thus the problem we consider is to design a compact fault-tolerant sourcewise *approximate* distance oracle.

Recall that "sourcewise" captures the fact that we are interested in distances between pairs $(s, v) \in S \times V$. For any pair $(s, v) \in S \times V$ and $f \in E$, let $||sv \diamond f||$ denote the distance from s to v in $G - f$. A fault-tolerant approximate distance oracle is said to have *multiplicative stretch* α if the distance $d_{G-f}(s, v)$ returned by the oracle on query $Qu(s, v, f)$ is sandwiched between the actual distance and α times the actual distance, i.e., $||sv \diamond f|| \leq d_{G-f}(s, v) \leq \alpha \cdot ||sv \diamond f||$. We show the following result.

▶ **Theorem 1.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 5 and size $\widetilde{O}(|S|n + n^{3/2})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

Note that our oracle has size $\widetilde{O}(n^{3/2})$ when $|S| = O(\sqrt{n})$. For smaller sets S, we show a sparser fault-tolerant sourcewise approximate distance oracle at the expense of a larger stretch. Its query answering time is also $O(1)$.

▶ **Theorem 2.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 13 and size $\widetilde{O}(|S|n + n^{4/3})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

Thus the above oracle has size $\widetilde{O}(n^{4/3})$ when $|S| = O(n^{1/3})$. The work of Bilò, Gualà, Leucci, and Proietti on multiple-edge fault-tolerant approximate shortest path trees [11] in undirected weighted graphs with a *single source* (so $|S| = 1$) implies a multiple-edge fault-tolerant *sourcewise*[1] (so S is any subset of V) approximate distance oracle of size $\widetilde{O}(|S|n)$ with a stretch of 3. Thus their oracle is sparser than our oracles when $|S|$ is small and it also achieves a better stretch. However, our query answering time is $O(1)$ while theirs is $O(\log^2 n)$, where n is the number of vertices. Our algorithms are truly simple while their techniques are quite involved.

As mentioned above, the problem of constructing fault-tolerant sourcewise exact distance oracles in undirected unweighted graphs has been studied earlier. Also, in undirected weighted graphs, the problem of constructing fault-tolerant *single source* (so $|S| = 1$) exact distance oracles has been studied. We discuss these results below.

Background. The first fault-tolerant exact distance oracle was designed by Demetrescu and Thorup in 2002 [14] and it was for directed weighted graphs. Their oracle handles single edge failures and has size $O(n^2 \log n)$ with $O(1)$ query time. After this result, there has

[1] Unfortunately, we were unaware of this work at the time of paper submission. We thank Manoj Gupta for bringing this paper to our attention.

been a long line of research on the problem of efficiently constructing single edge/vertex fault-tolerant exact distance oracles. Ignoring preprocessing time, the most space-efficient oracle is by Duan and Zhang [19] with size $O(n^2)$ and query time $O(1)$. Thus it shaves off the $\log n$ factor from the size of the original oracle.

Fault-tolerant sourcewise distance oracles. For undirected unweighted graphs, Gupta and Singh [20] designed a single edge fault-tolerant sourcewise exact distance oracle of size $\tilde{O}(n^{3/2}\sqrt{|S|})$ with $\tilde{O}(1)$ query time and source set S. In undirected graphs with edge weights in the range $\{1, 2, \ldots, M\}$, Bilò, Cohen, Friedrich and Schirneck [10] designed a fault-tolerant single source exact distance oracle. This oracle handles single edge failures and has size $\tilde{O}(n^{3/2}\sqrt{M})$ with query time $\tilde{O}(1)$. For undirected unweighted graphs, Dey and Gupta [15] designed a different oracle with the same space and query time bounds as in [10], but with a faster preprocessing time.

ST-distance oracles in directed graphs. In directed weighted graphs, Bilò, Choudhary, Gualà, Leucci, Parter and Proietti [9] designed a fault-tolerant ST-distance oracle, i.e., it maintains exact distances for all pairs in $S \times T$, for given vertex subsets S and T. It handles single edge failures and has size $\widetilde{O}((|S| + |T|)n)$ with $O(1)$ query time, where n is the number of vertices. They also designed a fault-tolerant ST-distance oracle in unweighted directed graphs of size $\widetilde{O}(n\sqrt{|S||T|})$ with query time $O(\sqrt{|S||T|})$. Furthermore, they showed a fault-tolerant *ST-approximate* distance oracle in directed unweighted graphs that returns in constant time a distance estimate stretched by an additive term. In particular, when $|S| = O(\sqrt{n})$, their oracle has size $\widetilde{O}(n^{3/2})$ and additive stretch $\widetilde{O}(\sqrt{n})$.

Fault-tolerant approximate distance oracles. Approximate distance oracles that provide distances within a small multiplicative stretch for all vertex pairs have been extensively studied. Table 1 summarizes results for fault-tolerant approximate distance oracles in directed/undirected graphs. Note that the stretch here is multiplicative, except for the last row where the stretch has an additive term as well.

Table 1 A table listing the works related to fault-tolerant approximate distance oracles where D is the diameter of the graph.

Graph	Faults	Stretch	Size	Query time	Ref
Undirected Weighted	$c \geq 1$	$(8k-2)(c+1)$, $k \geq 1$ integer	$O(ckn^{1+1/k}\log(nM))$, M is the max edge wt	$\tilde{O}(c)$	[13]
Undirected Unweighted	$c = 1$	$(2k-1)(1+\epsilon)$, $k \geq 1$ integer and $\epsilon > 0$	$\tilde{O}\left(\frac{k^5}{\epsilon^4}n^{1+1/k}\right)$	$O(1)$	[2]
Undirected Weighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(1+\epsilon)$	$O(n^2(\log D/\epsilon)^c c\log D)$	$O(c^5\log D)$	[12]
Directed Unweighted	$c \geq 2$	$(3+\epsilon)$	$\tilde{O}(n^{2-\frac{\alpha}{c+1}}/\epsilon)(\log n/\epsilon)^c)$ where $\alpha \in (0, 1/2)$ and $\epsilon > 0$	$O(n^\alpha/\epsilon^2)$	[6]
Undirected Weighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(2k-1)$ where $k \geq 1$ integer	$O(n^{1+\frac{1}{k}+\alpha+o(1)})$ where $\alpha \in (0, 1)$	$O(n^{1+\frac{1}{k}-\frac{\alpha}{k(c-1)}})$	[8]
Undirected Unweighted	$c = o\left(\frac{\log n}{\log\log n}\right)$	$(\frac{k+1}{k})(1+\epsilon)$ with additive stretch of 2, $k \geq 1$ integer and $\epsilon \geq 0$	$O\left(\frac{n^{2-\frac{\gamma}{(k+1)(c+1)}+o(1)}}{\epsilon^{c+2}}\right)$ where $\gamma \in \left(0, \frac{k+1}{2}\right)$	$O(n^\gamma/\epsilon^2)$	[7]

For single edge faults, note that Chechik, Langberg, Peleg, and Roditty [13] showed an approximate distance oracle with stretch 12 and size $\widetilde{O}(n^{3/2})$ and another with stretch 28 and size $\widetilde{O}(n^{4/3})$. In comparison to this, Theorem 1 shows a sourcewise approximate distance oracle with stretch 5 and size $\widetilde{O}(|S|n+n^{3/2})$ and Theorem 2 shows a sourcewise approximate distance oracle with stretch 13 and size $\widetilde{O}(|S|n+n^{4/3})$. Thus for small sets S, our oracles are as sparse and have smaller stretch. Note that for single faults and every $k \geq 1$, approximate distance oracles by Baswana and Khanna [2] are almost as sparse as the oracles in [13] and have significantly smaller stretch. However these oracles work only for unweighted graphs.

It is an open problem if our construction can be generalized to work for all integers k, in other words, to show a sourcewise approximate distance oracle of size $\widetilde{O}(|S|n+n^{1+1/k})$ and stretch $8k-3$ with $O(1)$ query answering time for $k \geq 3$. Our results show such a construction for $k = 1, 2$. Note that the remaining approximate distance oracles in Table 1 have superconstant query time, so our oracles cannot directly be compared with them.

Our techniques. Our algorithms are simple to describe and use the ST-distance oracle by Bilò, Choudhary, Gualà, Leucci, Parter and Proietti [9]. Their oracle uses *landmark* vertices, i.e., vertices picked uniformly at random from the vertex set V (originally used by Bernstein and Karger [4]).[2] The oracle in Theorem 1 uses this ST-distance oracle for the given source set S and $T = S \cup \mathcal{L}$, where $\mathcal{L}$ is our landmark vertex set. The oracle in Theorem 2 is based on the same idea, however there are two levels of sampling here: so we have two landmark vertex sets $\mathcal{L}_2 \subseteq \mathcal{L}_1$. Theorem 1 and Theorem 2 are proved in Section 3 and Section 4, respectively. We discuss preliminaries in Section 2 and conclude in Section 5.

2 Preliminaries

This section describes the notation that will be used in the rest of the paper and also gives a sketch of the ST-distance oracle from [9]. Our input is an undirected graph $G = (V, E)$ with positive edge weights as given by $\mathsf{wt} : E \to \mathbb{R}_+$. For any path ρ in G:

- let $||\rho||$ be the *length* of ρ, i.e., $||\rho|| = \sum_{e \in \rho} \mathsf{wt}(e)$;
- let $|\rho|$ be the *hop length* of ρ, i.e., the number of edges in ρ.

For any $(u, v) \in V \times V$, a shortest path between u and v is a path of minimum length between u and v. We assume the shortest path between any two vertices in the graph is unique. This property can be achieved by random perturbation of the given edge weights (e.g., see [22]). The property of unique shortest paths was also used in [5, 17, 18, 20, 21]. We denote the shortest path from u to v by uv. Thus $||uv||$ is the distance between u and v in G and $|uv|$ is the hop length between u and v in G.

- Let $G - f = (V, E \setminus \{f\})$ be the graph obtained after deleting edge f from the graph G. As in G, we assume there is a unique shortest path between any pair of vertices in $G - f$.
- For any $(u, v) \in V \times V$ and $f \in E$, let $uv \diamond f$ be the shortest path between u and v in $G - f$. So $||uv \diamond f||$ is the distance between u and v in $G - f$.

The concept of *landmark* vertices will be key to our distance oracles.

▶ **Definition 3** (Landmark Vertex Set, $\mathcal{L}$)**.** *Sample each vertex in G independently with probability p. The selected set (call it $\mathcal{L}$) of vertices is the landmark vertex set.*

[2] To the best of our knowledge, the name "landmark" vertices was first used by Dey and Gupta [16].

The probability p in Definition 3 will be set to different values in Section 3 and Section 4. The following proposition on the landmark vertex set $\mathcal{L}$ will be very useful to us.

▶ **Proposition 4.** *With high probability, for any pair of vertices u and v, if $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$ then there is at least one landmark vertex on uv.*

Proof. Since each vertex in G is sampled independently with probability p, for any pair of vertices u and v, the probability that there is *no* landmark vertex on uv is $(1-p)^k$ where $k = |uv| + 1$ is the number of vertices on uv. Because $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$, the probability that there is no landmark vertex on uv is at most:

$$(1-p)^{\frac{3 \ln n}{p}} \leq \left(\frac{1}{e}\right)^{3 \ln n} \leq \frac{1}{n^3}.$$

Thus for any pair of vertices u and v with $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$, the probability that there is no landmark vertex on uv is at most $1/n^3$. Hence the probability that there is some pair $(x, y) \in V \times V$ with $|xy| \geq \lfloor \frac{3 \ln n}{p} \rfloor$ such that there is no landmark vertex on xy is at most $\binom{n}{2}/n^3 \leq 1/n$. Thus with probability at least $1 - 1/n$, it is the case that for every pair $(u, v) \in V \times V$ with $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$, there is at least one landmark vertex on uv. ◀

Since each vertex in G is sampled with probability p, the expected size of $\mathcal{L}$ is np. We will set $p = n^{-\delta}$ for some $\delta \in (0, 1)$ in Section 3 and Section 4. Thus with high probability, we will have $|\mathcal{L}| \leq 2np$ (by Chernoff bound).

- If either $|\mathcal{L}| > 2np$ or there exists a pair of vertices u, v with $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$ such that there is no vertex of $\mathcal{L}$ on uv then we will repeat the step of sampling vertices and construct another landmark vertex set such that both these properties hold for the set obtained.

Thus we will assume that $|\mathcal{L}| = O(np)$ and every pair of vertices u, v with $|uv| \geq \lfloor \frac{3 \ln n}{p} \rfloor$ has at least one vertex of $\mathcal{L}$ on uv. The expected number of trials to obtain a desired landmark set $\mathcal{L}$ is $O(1)$.

Fault-Tolerant ST-Distance Oracle. We now briefly discuss the algorithm of Bilò et al.[9] to construct a fault-tolerant exact distance oracle in G for pairs $(s, t) \in S \times T$, where $S \subseteq V$ and $T \subseteq V$ are part of the input. Fix a pair $(s, t) \in S \times T$ and let $f = (a, b)$ be any edge on st. Let ℓ and ℓ' be the two landmark vertices on st closest to a and b on as and bt, respectively.

There are 3 cases with respect to the replacement path $st \diamond f$: (i) $st \diamond f$ goes through ℓ, (ii) $st \diamond f$ goes through ℓ', (iii) $st \diamond f$ goes through neither ℓ nor ℓ'. Their algorithm builds tables to deal with each of these cases. Figure 1 captures the main idea.

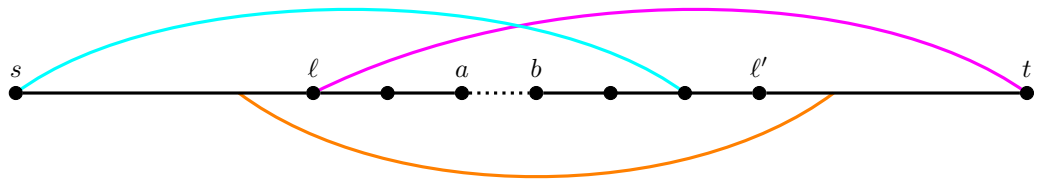


Figure 1 The replacement path $st \diamond f$ where $f = (a, b)$ in case (i) is $s\ell$ followed by the magenta path $\ell t \diamond f$; in case (ii) it is the blue path $s\ell' \diamond f$ followed by $\ell' t$ and in case (iii) $st \diamond f$ avoids both ℓ and ℓ' - so the orange path is part of $st \diamond f$.

The following theorem from [9] will be used in our algorithms.

▶ **Theorem 5** ([9]). *An n-vertex directed or undirected weighted graph G for given subsets S and T of V can be preprocessed in polynomial time to compute a data structure of size $O((|S|+|T|)n\log n)$ that given any pair $(s,t)\in S\times T$ and any failing edge f can report $||st\diamond f||$ in constant time.*

3 Sourcewise Approximate Distance Oracle for a Single Edge Fault

Our input is an undirected weighted graph $G=(V,E)$ with a positive weight function $\mathsf{wt}:E\to\mathbb{R}_+$ and a subset $S\subseteq V$ of sources. The goal is to build a compact data structure that can answer distance queries $Qu(s,v,f)$ within a small multiplicative stretch, where $s\in S, v\in V$ and f is the edge fault.

Landmark vertex set $\mathcal{L}$. Recall Definition 3 on landmark vertices. Let us sample each vertex independently with probability $p=(3\ln n)/\sqrt{n}$ to obtain our landmark vertex set $\mathcal{L}$. The following two properties hold (see Proposition 4); otherwise we resample to obtain another landmark vertex set $\mathcal{L}$ so that the following two properties hold.

- $|\mathcal{L}|=O(\sqrt{n}\log n)$.
- For any pair of vertices u and v: if $|uv|\geq\lfloor\sqrt{n}\rfloor$, then there is at least one landmark vertex on uv.

Our algorithm. On input $G=(V,E)$ and $S\subseteq V$, the first step of our algorithm is to build the above landmark vertex set $\mathcal{L}$. We then compute shortest path trees $\mathcal{T}(u)$ rooted at u for all $u\in S\cup\mathcal{L}$. Along with every vertex v, the tree $\mathcal{T}(u)$ also has the two attributes $||uv||$ and $|uv|$, i.e., the length and the hop length of uv.

Our algorithm constructs the ST-exact distance oracle from [9] fixing the source set S and destination set $T=S\cup\mathcal{L}$. For each $v\in V$, let t_v be the vertex in T that is closest to v, where ties are broken arbitrarily. We maintain the lengths of replacement paths $vt_v\diamond f$ for each edge $f\in vt_v$. Our algorithm is described below.

1. Obtain the landmark vertex set $\mathcal{L}$.
2. For each $u\in S\cup\mathcal{L}$ do: compute the shortest path tree $\mathcal{T}(u)$ rooted at u in $G=(V,E)$.
3. Use Theorem 5 to construct an ST-exact distance oracle for the given source set S and target set $T=\mathcal{L}\cup S$ in $G=(V,E)$.
4. For every $v\in V$ in the graph $G=(V,E)$ do:
   - Identify the nearest vertex to v in the target set $T=\mathcal{L}\cup S$. Call this vertex t_v.
   - For $1\leq i\leq|vt_v|$ do:
     - Let f_i be the i-th edge from t_v on vt_v.
     - Compute the distance $||vt_v\diamond f_i||$ between v and t_v in $G-f_i$.
     - Set $\mathsf{Dist}_T[v,i]=||vt_v\diamond f_i||$.

Query answering algorithm. In response to the query $Qu(s,v,f)$, the query answering algorithm first checks if $f\in sv$. This check can be done efficiently via LCA queries. Given a rooted tree $\mathcal{T}$ and a pair of vertices x,y in the tree $\mathcal{T}$, recall that $\mathsf{LCA}_{\mathcal{T}}(x,y)$ is the least common ancestor of x and y in tree $\mathcal{T}$.

Observe that $f=(a,b)\in sv$ if and only if the answer to the following three questions is "yes" where $\mathcal{T}(s)$ is the shortest path tree in G rooted at s.

- Is $\mathsf{LCA}_{\mathcal{T}(s)}(v,a)$ equal to a?
- Is $\mathsf{LCA}_{\mathcal{T}(s)}(v,b)$ equal to b?
- Is $|sa|+1=|sb|$ or is $|sb|+1=|sa|$?

A "yes" answer to the first two questions implies that both a and b are vertices on the path sv. Moreover, a and b are adjacent to each other on sv if and only if the answer to the third question is "yes". Recall that for any vertex w, $|sw|$ is the hop length between s and w, i.e., the number of edges in sw.

Given a tree $\mathcal{T}$, there is a linear time algorithm to build an $O(n)$ size data structure such that LCA queries on $\mathcal{T}$ can be answered in $O(1)$ time [3]. Recall that for every vertex w, the hop length $|sw|$ is stored along with w in $\mathcal{T}(s)$. Thus $|sa|$ and $|sb|$ can be retrieved in $O(1)$ time. Hence the query answering algorithm can determine in $O(1)$ time if $f \in sv$ or not. The query answering algorithm will return $||sv||$ if $f \notin sv$ (see Figure 2). Recall that the distance $||sv||$ is also stored along with v in $\mathcal{T}(s)$.



Figure 2 Here $f = (a, b) \notin sv$, so the path sv is undisturbed by the edge fault f.

If $f \in sv$, then the query answering algorithm looks up the identity of t_v, which is the nearest vertex in T to v. Via LCA queries on $\mathcal{T}(t_v)$, we can determine if $f = (a, b) \in vt_v$ or not. If not, then $||vt_v \diamond f|| = ||vt_v||$. So let us assume $f \in vt_v$.

Assume without loss of generality that b is closer than a to t_v, i.e., $|at_v| = |bt_v| + 1$ (see Figure 2). Let i be the index such that a is the i-th vertex from t_v on vt_v. Then $||vt_v \diamond (a, b)|| = \mathsf{Dist}_T[v, i]$. Recall that the attribute $i = |at_v|$ is stored along with a in $\mathcal{T}(t_v)$.

- The query answering algorithm returns $||vt_v \diamond f|| + ||st_v \diamond f||$, where $||vt_v \diamond f|| = \mathsf{Dist}_T[v, i]$.

Note that the distance $||st_v \diamond f||$ is obtained by querying the ST-distance oracle. Thus the query answering time is $O(1)$. We show below that $||vt_v \diamond f|| + ||st_v \diamond f|| \leq 5||sv \diamond f||$.

▶ **Lemma 6.** *For any $(s, v) \in S \times V$ and $f \in E$, our algorithm returns an estimate for the s-v distance in $G - f$ with a multiplicative stretch of at most 5 in constant time.*

Proof. It follows from the discussion above that the query answering time is $O(1)$. Since the query answering algorithm will return $||sv||$ if $f \notin sv$ (see Figure 2), let us assume $f \in sv$. Then the distance estimate returned by the query answering algorithm in response to query $Qu(s, v, f)$ is $||st_v \diamond f|| + ||vt_v \diamond f||$ where t_v is the nearest vertex in T to v. We now bound the sum $||st_v \diamond f|| + ||vt_v \diamond f||$. Consider the following four cases.

1. $f \in st_v$ and $f \in vt_v$. This means the edge f belongs to the shortest path between t_v and the least common ancestor of s and v in $\mathcal{T}(t_v)$ (see Figure 2). However then $f \notin sv$, contradicting our assumption that $f \in sv$.
2. $f \in st_v$ and $f \notin vt_v$. The query answering algorithm will determine via LCA queries that $f \notin vt_v$, so $||vt_v \diamond f|| = ||vt_v||$. Let us bound $||st_v \diamond f||$. The graph $G - f$ has an s-t_v path obtained by stitching the paths $sv \diamond f$ and vt_v, i.e., the path $sv \diamond f$ followed by vt_v. So $||st_v \diamond f|| \leq ||sv \diamond f|| + ||vt_v||$. Hence the distance returned is at most $||sv \diamond f|| + 2||vt_v||$.
   - Because t_v is the closest vertex in $T = \mathcal{L} \cup S$ to v, we have $||vt_v|| \leq ||vs||$. Hence the distance returned is at most $||sv \diamond f|| + 2||sv|| \leq 3||sv \diamond f||$. So the stretch is at most 3 in this case.

3. $f \notin st_v$ and $f \in vt_v$. Since $f \notin st_v$, we have $||st_v \diamond f|| = ||st_v||$. Let us bound $||vt_v \diamond f||$. Since the graph $G - f$ has a vt_v path obtained by stitching $sv \diamond f$ and st_v, we have $||vt_v \diamond f|| \leq ||sv \diamond f|| + ||st_v||$ (see Figure 3). Thus the distance returned by the oracle is at most $||sv \diamond f|| + 2||st_v||$.
   - Observe that $||st_v|| \leq ||sv|| + ||vt_v|| \leq 2||sv||$. Hence the distance returned is at most $||sv \diamond f|| + 4||sv|| \leq 5||sv \diamond f||$. Thus the stretch is at most 5 in this case.

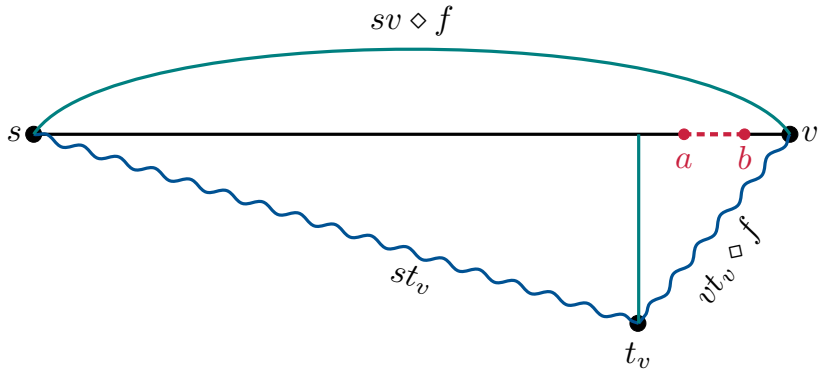


Figure 3 The oracle returns a distance estimate $\leq 2||st_v|| + ||sv \diamond f||$, so the stretch is ≤ 5.

4. $f \notin st_v$ and $f \notin vt_v$. The query answering algorithm will return $||st_v \diamond f|| + ||vt_v \diamond f|| = ||st_v|| + ||vt_v||$ in this case. We have $||vt_v|| \leq ||vs||$ and we also have $||st_v|| \leq ||sv|| + ||vt_v|| \leq 2||sv||$. Thus the stretch is at most 3 in this case.

This finishes the proof of the lemma. ◀

Data structures constructed. Our algorithm constructs in step 3 all the data structures constructed by the ST-distance oracle algorithm. Thus we have access to $||st \diamond f||$ for every $(s,t) \in S \times T$ and $f \in E$. Our oracle also has the table Dist_T that stores $||vt_v \diamond f||$ between v and t_v in $G - f$, for each $v \in V$ and edge $f \in vt_v$. Let us bound the size of our oracle.

▶ **Lemma 7.** *The size of the data structures constructed by our algorithm is* $\widetilde{O}(|S|n + n^{3/2})$.

Proof. The size of T is $\widetilde{O}(|S| + \sqrt{n})$. So the sizes of all the shortest path trees constructed in step 2 is $\widetilde{O}(|S|n + n^{3/2})$. Similarly, the size of the ST-oracle constructed in step 2 is $\widetilde{O}(|S|n + n^{3/2})$ (by Theorem 5). Furthermore, the data structure used to answer LCA queries on each shortest path tree $\mathcal{T}(u)$ has size $O(n)$. Since $u \in S \cup \mathcal{L}$, these data structures also take up space $\widetilde{O}(|S|n + n^{3/2})$.

It follows from the property of our landmark set $\mathcal{L}$ that for any vertex v, we have $\min\{|v\ell| : \ell \in \mathcal{L}\} \leq \lfloor\sqrt{n}\rfloor$. So for each vertex v, we have $||vt_v \diamond f||$ stored for at most $\lfloor\sqrt{n}\rfloor$ many edges f, where t_v is the nearest vertex in T to v. Thus the size of Dist_T is $O(n \cdot \sqrt{n}) = O(n^{3/2})$. This finishes the proof of the lemma. ◀

It is easy to see that our algorithm runs in expected polynomial time. Recall that we used randomization to construct the set $\mathcal{L}$. By blowing up the size of $\mathcal{L}$ by a factor of $\log n$, the construction of $\mathcal{L}$ can be made deterministic (see [9, Lemma 1]). Thus Theorem 1 follows. We restate it below for convenience.

▶ **Theorem 1.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 5 and size $\widetilde{O}(|S|n + n^{3/2})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

▶ Remark 8. The query answering algorithm can return not only an approximate estimate of the distance $||sv \diamond f||$, but also the corresponding approximate shortest path in a succinct form. It is known that any replacement path $\rho \diamond f$ is *2-decomposable*, i.e., it is a concatenation of at most 2 shortest paths interleaved with at most 1 edge [1].

So along with any distance $||st_v \diamond f||$ (similarly, $||vt_v \diamond f||$), we could also store the corresponding replacement paths in 2-decomposable form. Thus the query answering algorithm can return the corresponding s-v approximate shortest path as the union of two replacement paths $\rho_1 = st_v \diamond f$ and $\rho_2 = vt_v \diamond f$, each in 2-decomposable form, say, $\rho_1 = \langle s, x, y, t_v \rangle$ and $\rho_2 = \langle v, x', y', t_v \rangle$. This will mean ρ_1 is the shortest path in G between s and x followed by the edge (x, y), and the shortest path in G between y and t_v, similarly for ρ_2.

4 A Sparser Fault-Tolerant Sourcewise Approximate Distance Oracle

In this section, we present another fault-tolerant sourcewise approximate distance oracle for single edge faults. As before, the input is an undirected weighted graph $G = (V, E)$ with a positive weight function $\mathsf{wt} : E \to \mathbb{R}_+$ and a subset $S \subseteq V$ of sources. Our goal is to build a sparser data structure that can answer distance queries $Qu(s, v, f)$ within a small multiplicative stretch. For sets S of size $o(\sqrt{n})$, the oracle in this section will be sparser than the one in Section 3.

Our algorithm. We will now construct *two* sets $\mathcal{L}_1$ and $\mathcal{L}_2$ of landmark vertices. We will first run the sampling step in Section 2 with $p = (3 \ln n)/n^{1/3}$. Let $\mathcal{L}_1$ be the resulting landmark set. The following properties follow from Section 2 (see Proposition 4).

- $|\mathcal{L}_1| = O(n^{2/3} \log n)$.
- For any pair of vertices u and v: if $|uv| \geq \lfloor n^{1/3} \rfloor$ then there is at least one vertex of $\mathcal{L}_1$ on uv.

After that, we sample each vertex of $\mathcal{L}_1$ with probability $1/n^{1/3}$. Let $\mathcal{L}_2$ be the set of selected vertices. Observe that this 2-step sampling to obtain $\mathcal{L}_2$ is equivalent to running the sampling step in Section 2 with $p = (3 \ln n)/n^{2/3}$ on the entire vertex set V. Thus the following properties follow from Section 2 (see Proposition 4).

- $|\mathcal{L}_2| = O(n^{1/3} \log n)$.
- For any pair of vertices u and v: if $|uv| \geq \lfloor n^{2/3} \rfloor$ then there is at least one vertex of $\mathcal{L}_2$ on uv.

Rather than sampling each vertex of V with probability $p = (3 \ln n)/n^{2/3}$ to get $\mathcal{L}_2$, we did this in two steps so that we have $\mathcal{L}_2 \subseteq \mathcal{L}_1$. Let $T_1 = \mathcal{L}_1 \cup S$ and let $T_2 = \mathcal{L}_2 \cup S$. Our algorithm will use the following notations for any vertex v.

- Let t_v be the vertex in T_1 that is nearest to v.
- Let t'_v be the vertex in T_2 that is nearest to v.

For each $u \in T_2$, we will keep the shortest path tree $\mathcal{T}(u)$ in G rooted at u. However, we cannot afford to keep shortest path trees rooted at each $u \in \mathcal{L}_1$, since that would exceed the desired space bound. Corresponding to each $u \in \mathcal{L}_1$, let $\mathsf{Ball}(u) = \{v \in V : t_v = u\}$ be the set of all vertices v that regard u as their nearest vertex in T_1.

- For each $v \in \mathsf{Ball}(u)$, we will store the path uv.
- Thus we keep a *truncated* shortest path tree $\hat{\mathcal{T}}(u) = \cup_{v \in \mathsf{Ball}(u)} uv$ in G rooted at u for each $u \in \mathcal{L}_1$.

Along with each vertex $v \in \hat{\mathcal{T}}(u)$ where $u \in \mathcal{L}_1$, we also store $|uv|$, i.e., the hop length of uv, and the distance $||uv||$. Similarly, as done in Section 3, along with each vertex $v \in \mathcal{T}(u)$, where $u \in T_2$, we store $|uv|$ and $||uv||$.

Below we describe the steps in our algorithm.

1. Obtain the landmark sets $\mathcal{L}_1$ and $\mathcal{L}_2$, where $\mathcal{L}_2 \subseteq \mathcal{L}_1$, as described above.
2. For each $u \in T_2 = S \cup \mathcal{L}_2$ do: compute the shortest path tree $\mathcal{T}(u)$ rooted at u in G.
3. Use Theorem 5 to construct an ST-exact distance oracle for the given source set S and target set $T = T_2$ in $G = (V, E)$.
4. For each $u \in \mathcal{L}_1$ do: compute the *truncated* shortest path tree $\hat{\mathcal{T}}(u)$ rooted at u in G.
5. For every $v \in V$ do:
   a. Let $t_v \in T_1 = S \cup \mathcal{L}_1$ be the vertex in T_1 that is nearest to v.
   b. For $1 \leq i \leq |vt_v|$ do:
      - Set $\mathsf{Dist}_1[v, i] = ||vt_v \diamond f_i||$ where f_i is the i-th edge from t_v on vt_v.
6. For every $u \in T_1$ do:
   a. Let $t'_u \in T_2$ be the vertex in T_2 that is nearest to u.
   b. For $1 \leq j \leq |ut'_u|$ do:
      - Set $\mathsf{Dist}_2[u, j] = ||ut'_u \diamond f_j||$ where f_j is the j-th edge from t'_u on ut'_u.

Observe that the array $\mathsf{Dist}_1[v, i]$ stores for any vertex v, the distance $||vt_v \diamond f||$ where f is the i-th edge from t_v on the path vt_v. Similarly, the array $\mathsf{Dist}_2[u, j]$ stores for any vertex $u \in T_1$, the distance $||ut'_u \diamond f||$ where f is the j-th edge from t'_u on the path ut'_u.

The query answering algorithm. In response to the query $Qu(s, v, f)$, the query answering algorithm first checks if $f \in sv$. As described in Section 3, this is done by checking the answers to some LCA queries in $\mathcal{T}(s)$. Let us assume $f \in sv$, otherwise the query answering algorithm will return $||sv||$.

Then the query answering algorithm looks up $x = t_v$ and $y = t'_x$. In more detail, (i) x is the closest vertex to v in T_1 and (ii) y is the closest vertex to x in T_2. The query answering algorithm needs to know if $f \in vx$ or not; if so, it also needs to know the index $i \in \{1, \ldots, n^{1/3}\}$ such that f is the i-th edge on xv. As described in Section 3, we can decide if $f \in vx$ or not via LCA queries on the truncated shortest path tree $\hat{\mathcal{T}}(x)$. If so, we can also obtain from $\hat{\mathcal{T}}(x)$ the value i such that f is the i-th edge from x on xv.

Thus the query answering algorithm knows in $O(1)$ time whether $f \in xv$ or not and if so, the index i such that f is the i-th edge from x on xv.

- If $f \notin vx$ then $||vx \diamond f|| = ||vx||$; else $||vx \diamond f|| = \mathsf{Dist}_1[v, i]$.

Recall that we compute $\mathcal{T}(u)$ for all $u \in T_2$. Thus, as described in Section 3, we can efficiently check if $f \in xy$ or not; if so, the algorithm also knows the index j such that f is the j-th edge from y on the path xy.

- If $f \notin xy$ then $||xy \diamond f|| = ||xy||$; else $||xy \diamond f|| = \mathsf{Dist}_2[x, j]$.

Since $s \in S$ and $y \in T_2$ (recall that $T = T_2$), the distance $||ys \diamond f||$ is obtained by querying the ST-distance oracle. Thus the query answering algorithm can obtain $||vx \diamond f||$, $||xy \diamond f||$, and $||ys \diamond f||$ in $O(1)$ time. In response to the query $Qu(s, v, f)$, the query answering algorithm returns $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$.

We will show in Lemma 9 that our s-v distance estimate in $G - f$ is at most $13||sv \diamond f||$.

▶ **Lemma 9.** *For any $(s, v) \in S \times V$ and $f \in E$, our algorithm returns an s-v distance estimate with stretch ≤ 13 in $G - f$ in $O(1)$ time.*

Proof. Suppose the query is $Qu(s, v, f)$. If $f \notin sv$ then the algorithm returns $||sv||$, thus the stretch is 1 in this case. So assume $f \in sv$. Then the query answering algorithm returns $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$, where $x = t_v$ and $y = t'_x$. Let us bound the stretch.

We need to compare the sum $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$ with $||sv \diamond f||$. Let us first show the following claim.

▷ Claim 10. We have (i) $||vx|| \leq ||vs||$, (ii) $||xy|| \leq 2||vs||$, and (iii) $||ys|| \leq 4||vs||$.

Proof. It follows from the definition of $T_1 = \mathcal{L}_1 \cup S$ that both x and s are in T_1. Since x is the nearest vertex in T_1 to v, we have $||vx|| \leq ||vs||$. Recall that $y = t'_x$. Since y is the closest vertex in T_2 to x, we have $||xy|| \leq ||xt'_v||$, i.e., the x-y distance is at most the distance between x and t'_v (recall that t'_v is the nearest vertex in T_2 to v). Furthermore, $||xt'_v|| \leq ||xv|| + ||vt'_v||$.

Observe that both $||xv||$ and $||vt'_v||$ are at most $||sv||$ since $s \in T_1 \cap T_2$, so v's distance to its nearest vertex in T_1 and also in T_2 is at most $||sv||$. Thus $||xy|| \leq 2||sv||$. So we have $||ys|| \leq ||sv|| + ||vx|| + ||xy|| \leq ||sv|| + ||sv|| + 2||sv|| = 4||sv||$. ◁

We are now ready to bound $||vx \diamond f|| + ||xy \diamond f|| + ||ys \diamond f||$. There are 8 cases depending on the presence of edge f on various shortest paths.

1. $f \notin vx$ and $f \notin xy$ and $f \notin ys$. Then the algorithm returns $||vx|| + ||xy|| + ||ys||$.
   - It immediately follows from Claim 10 that the s-v distance estimate returned in this case is at most $7||sv|| \leq 7||sv \diamond f||$.

2. $f \notin vx$ and $f \notin xy$ and $f \in ys$. Then the algorithm returns $||vx|| + ||xy|| + ||ys \diamond f||$. Since the failed edge f belongs to neither vx nor xy, we have $||sy \diamond f|| \leq ||sv \diamond f|| + ||vx|| + ||xy||$. We have $||vx|| + ||xy|| \leq 3||sv||$ (by Claim 10).
   - Thus the s-v distance estimate returned in this case is at most $||sv \diamond f|| + 6||sv|| \leq 7||sv \diamond f||$ (by Claim 10).

3. $f \notin vx$ and $f \in xy$ and $f \notin ys$. Then the algorithm returns $||vx|| + ||xy \diamond f|| + ||ys||$. Observe that $G - f$ has an x-y path of length at most $||xv|| + ||vs \diamond f|| + ||sy||$. Since $||ys|| \leq 4||vs||$ (by Claim 10), this x-y path in $G - f$ is of length at most $||sv|| + ||sv \diamond f|| + 4||sv|| = 5||sv|| + ||sv \diamond f||$.
   - Using Claim 10 to bound $||vx||$ and $||ys||$, the s-v distance estimate returned in this case is at most $||sv|| + 5||sv|| + ||sv \diamond f|| + 4||sv|| = 10||sv|| + ||sv \diamond f|| \leq 11||sv \diamond f||$.

4. $f \in vx$ and $f \notin xy$ and $f \notin ys$. Then the algorithm returns $||vx \diamond f|| + ||xy|| + ||ys||$. Observe that $G - f$ has a v-x path of length at most $||vs \diamond f|| + ||sy|| + ||yx||$. This is of length at most $||sv \diamond f|| + 4||sv|| + 2||sv|| = 6||sv|| + ||sv \diamond f||$.
   - Using Claim 10 to bound $||xy||$ and $||ys||$, the s-v distance estimate returned in this case is at most $||sv \diamond f|| + 6||sv|| + 2||sv|| + 4||sv|| = 12||sv|| + ||sv \diamond f|| \leq 13||sv \diamond f||$.

5. $f \notin vx$ and $f \in xy$ and $f \in ys$. Consider the shortest path tree $\mathcal{T}(x)$ rooted at x in G. Since $f \notin vx$ and $f \in xy$, the edge $f \in wy$ where $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$. But the edge f also belongs to ys and sv – this is not possible (see Figure 4). Thus this case cannot arise.

6. $f \in vx$ and $f \notin xy$ and $f \in ys$. Consider the shortest path tree $\mathcal{T}(s)$ rooted at s in G and let $z = \mathsf{LCA}_{\mathcal{T}(s)}(v, x)$. Since $f \in sv$ and $f \in vx$, it follows that $f \in zv$. Hence $f \notin sx$. Thus there is a v-x path in $G - f$ of length $||vs \diamond f|| + ||sx||$. Since $||sx|| \leq ||sv|| + ||vx|| \leq 2||sv||$, this v-x path in $G - f$ has length $||sv \diamond f|| + 2||sv||$.

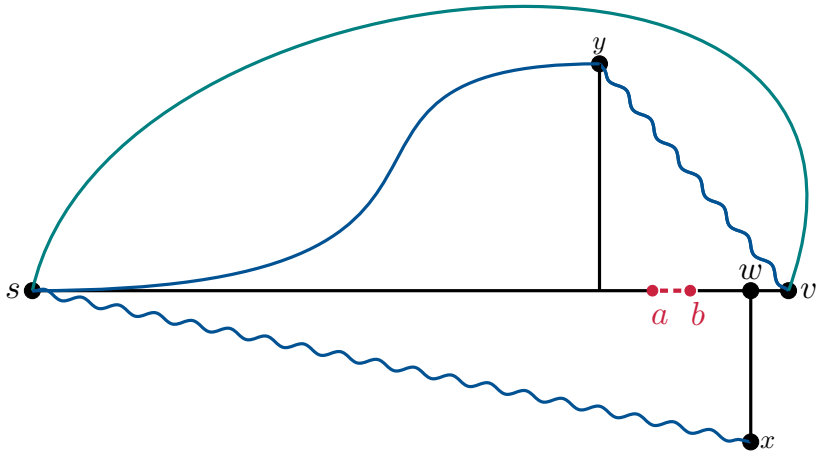


Figure 4 The edge $f = (a, b)$ is supposed to be in the paths sv, xy, and ys, but not in vx. Here $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$ where $\mathcal{T}(x)$ is the shortest path rooted at x in G.

Since $f \in sv$ and $f \in sy$, the edge $f \in sr$ where $r = \mathsf{LCA}_{\mathcal{T}(s)}(v, y)$. Thus the edge f does not belong to the path v-r-y in $\mathcal{T}(s)$. Hence there is an s-y path in $G - f$ of length $||sv \diamond f|| + ||vs|| + ||sy|| \leq ||sv \diamond f|| + ||vs|| + 4||sv||$ (by Claim 10).

- Thus $G - f$ has an s-y path of length $||sv \diamond f|| + 5||vs||$, plus a v-x path of length $||sv \diamond f|| + 2||vs||$. Since $||xy|| \leq 2||vs||$, the s-v distance estimate returned in this case is at most $2||sv \diamond f|| + 9||vs|| \leq 11||sv \diamond f||$.

7. $f \in vx$ and $f \in xy$ and $f \notin ys$. As seen in case 6, there is a v-x path in $G - f$ of length $||sv \diamond f|| + 2||vs||$. Moreover, since $f \in vx$ and $f \in xy$, the edge $f \in xw$ where $w = \mathsf{LCA}_{\mathcal{T}(x)}(v, y)$. Hence the edge f does not belong to the path v-w-y in $\mathcal{T}(x)$.

   Thus there is a v-y path in $G - f$ of length at most $||vx|| + ||xy|| \leq 3||sv||$. Hence there is an x-v-y path of length at most $||sv \diamond f|| + 2||vs|| + 3||sv|| = ||sv \diamond f|| + 5||vs||$.
   - So the s-v distance estimate returned in this case is at most $||sy|| + (||sv \diamond f|| + 5||vs||) + (||sv \diamond f|| + 2||vs||)$. Since $||sy|| \leq 4||sv||$, this is at most $2||sv \diamond f|| + 11||sv|| \leq 13||sv \diamond f||$.

8. $f \in vx$ and $f \in xy$ and $f \in ys$. As seen in case 7, there is a v-x path in $G - f$ of length $||sv \diamond f|| + 2||vs||$ and there is an x-y path of length at most $||sv \diamond f|| + 5||vs||$. It also follows from case 7 that there is a v-y path in $G - f$ of length at most $3||sv||$, thus there is an s-y path in $G - f$ of length at most $||sv \diamond f|| + 3||vs||$.
   - So the s-v distance estimate returned in this case is at most $3||sv \diamond f|| + 10||sv|| \leq 13||sv \diamond f||$.

Thus the stretch of our approximate distance oracle is at most 13. We have already seen that the query answering time is $O(1)$. This finishes the proof of the lemma. ◀

Size of the oracle. We show below in Lemma 11 that the space taken up by the data structures constructed in all the steps of our algorithm is $\widetilde{O}(n^{4/3} + |S|n)$.

▶ **Lemma 11.** *The space needed to store all the data structures constructed by our algorithm is* $\widetilde{O}(n^{4/3} + |S|n)$.

Proof. The space taken up by the truncated shortest path trees $\hat{\mathcal{T}}(u)$ for all $u \in \mathcal{L}_1$ is $O(\sum_{v \in V} |vt_v|)$. Observe that $|vt_v| \leq n^{1/3}$ (by Proposition 4). Thus $O(\sum_v |vt_v|) = O(n^{4/3})$. Similarly the space taken up by $\mathcal{T}(t)$ for all $t \in T_2$ is $O(n^{4/3} \log n + |S|n)$ since $|T_2| = |\mathcal{L}_2| + |S|$ and $|\mathcal{L}_2|$ is $O(n^{1/3} \log n)$. The size of the ST-oracle (where $T = T_2$) is also $\widetilde{O}(n^{4/3} + |S|n)$ (by Theorem 5).

For each vertex v, we store t_v and t'_v – these are the nearest vertices to v in T_1 and T_2, respectively. For all edges $f \in vt_v$, we store $||vt_v \diamond f||$ in the data structure Dist_1. We have $|vt_v| \leq n^{1/3}$ (by Proposition 4). Thus the space taken by the data structure Dist_1 to store the distances $\mathsf{Dist}_1[v, i]$ where $v \in V$ and $1 \leq i \leq n^{1/3}$ is at most $n^{4/3}$.

For all edges $f \in ut'_u$, where $u \in T_1$, the data structure Dist_2 stores $||ut'_u \diamond f||$. For any vertex u, we have $|ut'_u| \leq n^{2/3}$ (by Proposition 4). Thus the space taken by Dist_2 to store the distances $\mathsf{Dist}_2[u, i]$ where $u \in T_1$ and $1 \leq i \leq n^{2/3}$ is $|T_1| \cdot n^{2/3} = O((n^{2/3} \log n + |S|) \cdot n^{2/3})$, which is $O(n^{4/3} \log n + |S|n^{2/3})$. Thus the entire space taken up by all the data structures is $\widetilde{O}(n^{4/3} + |S|n)$. ◀

It is easy to see that our algorithm runs in expected polynomial time. As mentioned at the end of Section 3, by blowing up the sizes of $\mathcal{L}_1$ and $\mathcal{L}_2$ by a factor of $\log n$, their construction can be made deterministic as stated in [9, Lemma 1]. Moreover, we can easily ensure that $L_2 \subseteq L_1$. Thus Theorem 2 follows. We restate it below for convenience.

▶ **Theorem 2.** *Let $G = (V, E)$ be an undirected graph on n vertices with positive edge weights. For any $S \subseteq V$, a fault-tolerant sourcewise approximate distance oracle with multiplicative stretch at most 13 and size $\widetilde{O}(|S|n + n^{4/3})$ can be constructed in polynomial time such that $Qu(s, v, f)$ where $(s, v) \in S \times V$ and $f \in E$ can be answered in constant time.*

As mentioned in Remark 8, along with every distance in Dist_1 and Dist_2, we could also store the corresponding replacement paths in 2-decomposable form. Thus along with the approximate s-v distance, the query answering algorithm can also return the approximate path between s and v as the union of 3 paths, each in 2-decomposable form.

5 Concluding Remarks

Fault-tolerant approximate distance oracles that maintain approximate distances for all pairs of vertices have been well-studied. Fault-tolerant single source and multiple source *exact* distance oracles have also been studied. As mentioned in [9], given a subset $S \subseteq V$, for the problem of storing $||sv \diamond f||$ where $(s, v) \in S \times V$ and $f \notin E$ is allowed[3] (this is interpreted the same as if no edge has failed), using standard tools, it can be shown that there are n-vertex graph families, for which any representation that allows for the return of all the $S \times V$ post-failure distances must have size $\Omega(n^{3/2}\sqrt{|S|})$. This motivates the study of sparser data structures that maintain *approximate* distances for all pairs in $S \times V$ under the failure of any $f \in E$. Such a data structure is a fault-tolerant sourcewise approximate distance oracle.

We showed two such oracles: one of size $\widetilde{O}(|S|n + n^{3/2})$ and stretch 5 and another of size $\widetilde{O}(|S|n + n^{4/3})$ and stretch 13. The query time for both oracles is constant. Upon query $Qu(s, v, f)$ where $f \notin E$, it turns out that both our query answering algorithms return the original distance $||sv||$ as if no edge has failed. There are several interesting open problems:

- Are there approximate sourcewise distance oracles of size $\widetilde{O}(|S|n + n^{1+1/k})$ and stretch $8k - 3$ with $O(1)$ query answering time for all integers $k \geq 1$? Our constructions showed such oracles for $k = 1, 2$.

[3] We thank a reviewer for pointing out this subtlety to us.

- The study of fault-tolerant exact as well as approximate distance oracles has so far considered structured subsets of $V \times V$ such as $S \times T$. Is there a sparse fault-tolerant exact or approximate distance oracle for an arbitrary subset $\mathcal{P}$ of $V \times V$?

References

1. Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by path concatenation: fast recovery of mpls paths. *Distributed Computing*, 15(4):273–283, 2002. doi:10.1007/s00446-002-0080-6.
2. Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/S00453-012-9621-Y.
3. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
4. Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 34–43. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347087.
5. Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 101–110. ACM, 2009. doi:10.1145/1536414.1536431.
6. Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Approximate distance sensitivity oracles in subquadratic space. *TheoretiCS*, 3, 2024. doi:10.46298/THEORETICS.24.15.
7. Davide Bilò, Shiri Chechik, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Improved distance (sensitivity) oracles with subquadratic space. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 1550–1558. IEEE, 2024. doi:10.1109/FOCS61266.2024.00097.
8. Davide Bilò, Keerti Choudhary, Sarel Cohen, Tobias Friedrich, Simon Krogmann, and Martin Schirneck. Compact distance oracles with large sensitivity and low stretch. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures - 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 - August 2, 2023, Proceedings*, volume 14079 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2023. doi:10.1007/978-3-031-38906-1_11.
9. Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. 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 13:1–13:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.STACS.2018.13.
10. Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-optimal deterministic single-source distance sensitivity oracles. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.18.

11 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. *Algorithmica*, 84(1):37–59, 2022. URL: https://doi.org/10.1007/s00453-021-00879-8, doi:10.1007/S00453-021-00879-8.

12 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1 + \epsilon)$-approximate f-sensitive distance oracles. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1479–1496. SIAM, 2017. doi:10.1137/1.9781611974782.96.

13 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012. doi:10.1007/S00453-011-9543-0.

14 Camil Demetrescu and Mikkel Thorup. Oracles for distances avoiding a link-failure. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 838–843, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545490.

15 Dipan Dey and Manoj Gupta. Near optimal algorithm for fault tolerant distance oracle and single source replacement path problem. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.42.

16 Dipan Dey and Manoj Gupta. Near optimal dual fault tolerant distance oracle. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 45:1–45:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.45.

17 Dipan Dey and Manoj Gupta. Nearly optimal fault tolerant distance oracle. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 944–955. ACM, 2024. doi:10.1145/3618260.3649697.

18 Ran Duan and Hanlin Ren. Maintaining exact distances under multiple edge failures. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1093–1101. ACM, 2022. doi:10.1145/3519935.3520002.

19 Ran Duan and Tianyi Zhang. Improved distance sensitivity oracles via tree partitioning. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, volume 10389 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2017. doi:10.1007/978-3-319-62127-2_30.

20 Manoj Gupta and Aditi Singh. Generic single edge fault tolerant exact distance oracle. 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 72:1–72:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.72.

21 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959899.

22 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 779–790. Springer, 2013. doi:10.1007/978-3-642-40450-4_66.

Beyond Monads and Biproducts: A Uniform Interpretation of Parallelism in Intuitionistic Logic

Alejandro Díaz-Caro
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
Universidad Nacional de Quilmes, Buenos Aires, Argentina

Octavio Malherbe
IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

Abstract

Traditional approaches to modelling parallelism and algebraic structure in lambda calculi often rely on monads – as in Moggi's framework – or on rich categorical structures such as biproducts – as used in certain models of linear logic. In this work, we propose a minimal alternative that captures both parallelism and weighted parallelism (linear combinations) within the setting of intuitionistic propositional logic, without resorting to monads or assuming the existence of biproducts.

We introduce two lambda calculi: a parallel lambda calculus and an algebraic lambda calculus, both extending full propositional intuitionistic logic. Their semantics are given in two categories: $\mathbf{Mag}_{\mathbf{Set}}$, whose objects are magmas and arrows are functions in **Set**; and $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$, whose objects are action magmas.

The key technical challenge addressed is the interpretation of disjunction in the presence of parallel and algebraic operators. Since the usual coproduct structure is unavailable in our minimal setting, we propose a novel set-theoretic interpretation based on the union of the disjoint union and the Cartesian product. This allows for the construction of sound and adequate models for both calculi.

Our results offer a unified and structurally lightweight framework for modelling parallelism and algebraic effects in intuitionistic logic, opening the way to alternatives beyond the traditional monadic or linear logic approaches.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Type theory; Theory of computation → Categorical semantics; Theory of computation → Denotational semantics

Keywords and phrases Algebraic lambda calculus, Categorical semantics, Disjunction, Proof theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.28

Related Version *Full version with appendices*: `arXiv:2408.16102` [10]

Funding Supported by the European Union through the MSCA SE project QCOMICAL (Grant Agreement ID: 101182521) and by the Uruguayan CSIC grant 22520220100073UD.
Alejandro Díaz-Caro: Also supported by the Plan France 2030 through the PEPR integrated project EPiQ (ANR-21-PETQ-0007).

1 Introduction

Motivation. In his 1991 paper [18], Eugenio Moggi introduced a uniform framework to handle effectful computations in the lambda calculus, using monads to encapsulate various computational effects such as non-determinism, state, exceptions, and input/output. This monadic approach has since been widely adopted in functional programming languages, providing a powerful abstraction for reasoning about side effects. In the context of intuitionistic propositional logic – which corresponds categorically to cartesian closed categories – non-determinism can be captured, for instance, in the category **Set** by the map


© Alejandro Díaz-Caro and Octavio Malherbe;
licensed under Creative Commons License CC-BY 4.0

$\chi : A \times A \to \mathcal{P}A$ defined as $(x, y) \mapsto \{x, y\}$. This non-deterministic operator is sometimes referred to as a *parallel operator*, as it can be seen as preserving all possible outputs in parallel.

While this monadic framework is highly versatile, it does not extend seamlessly to all logical systems. In particular, within Intuitionistic Linear Logic (ILL), the monadic encoding of non-determinism becomes problematic: the map χ is not linear. For instance, $\chi(x_1 + x_2, y_1 + y_2) = \{x_1 + y_1, x_2 + y_2\}$, whereas $\chi(x_1, y_1) + \chi(x_2, y_2) = \{x_1, y_1\} + \{x_2, y_2\}$, which does not yield the same set under any reasonable definition of set addition.

An alternative approach, suitable for linear settings, was developed through a line of work on relational models. In particular, in [4], building on previous work developed in Manzonetto's doctoral thesis [16] and in [3], and in the context of untyped λ-calculus models, the hom-sets of the category **MRel**, restricted to a reflexive object, were enriched with a commutative semiring structure in order to model a parallel and non-deterministic λ-calculus. In the typed setting, this approach was pursued in weighted relational models of typed λ-calculi [14], where the effect is captured by the map $\chi' : A \times A \to A$ defined by $(x, y) \mapsto x + y$. In [11], this idea was extended into a more general categorical characterisation, relying on the presence of biproducts to define the sum.

The guiding question of the present work is the following: *Can we construct an alternative to Moggi's monad, inspired by the use of biproducts, that works in the context of propositional logic, where biproducts are not available?* We show that the answer is affirmative by providing a concrete model that captures this effect without requiring the full structure of a biproduct.

Our proposal is a structurally simple alternative capable of capturing both parallelism and weighted parallelism (or linear combinations). We move from the category **Set** to a richer setting based on magmas, where we define an ad hoc operation that mimics – to some extent – the behaviour of a biproduct, though it is not a biproduct in the categorical sense.

More precisely, we introduce the category $\mathbf{Mag}_{\mathbf{Set}}$, whose objects are magmas with no required algebraic properties, and whose arrows are simply functions from the category **Set**. In addition, we define the category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$, whose objects are *action magmas* – magmas equipped with an external scalar action – and whose arrows are again functions from **Set**. These constructions allow us to interpret the parallel lambda calculus and the algebraic lambda calculus, respectively, in a way reminiscent of the interpretation of the parallel linear lambda calculus in the category of vector spaces, but without requiring the existence of biproducts.

A key difficulty in adapting these techniques to intuitionistic propositional logic lies in the treatment of disjunction. In contrast to models of linear logic equipped with biproducts – where additive conjunction and disjunction are both interpreted via this common structure – the absence of a unified product-coproduct structure in propositional logic means that disjunction cannot be handled in the same way as conjunction. This asymmetry poses significant challenges in modelling the behaviour of parallel or algebraic combinations involving disjunctive terms. In this work, we address this difficulty directly by proposing a concrete interpretation of disjunction that preserves the intuition of additive structure, while remaining compatible with the categorical limitations of the setting.

Antecedents. Several foundational works have explored extensions of the lambda calculus to express superposition, linear combinations, and parallelism. The Lineal calculus [1] was introduced to represent quantum programs by incorporating the principle that, since data and programs are unified in the lambda calculus, the superposition of data implies the superposition of programs. This idea was formalised through a linear structure supporting algebraic combinations of terms.

An independent development is the Algebraic Lambda Calculus [19], a simplification of the Differential Lambda Calculus [13], which emphasises algebraic structure rather than operational semantics. Later, Lineal and the Algebraic Lambda Calculus were shown to be closely related [2].

Several semantic models have addressed non-determinism and parallelism. In [7], a filter model was proposed for a concurrent lambda calculus, where terms are interpreted as sets of possible outcomes. A relational model capturing non-deterministic and parallel behaviour was introduced in [5], and a general framework using weighted relations, allowing for deterministic, non-deterministic, and probabilistic computation, was proposed in [14].

More recently, the $\odot$-connective was introduced in [8] and extended in [9] to express quantum superpositions and measurements within a logical framework. These systems include additive structure in proof terms, such as sums and scalar products, aiming to represent quantum behaviour. A categorical interpretation for such calculi in a linear setting – using biproducts in the category of vector spaces – was developed in [11].

Our perspective. While Moggi's monadic approach provides a general and robust framework for handling computational effects in intuitionistic settings, our goal is to explore whether the structural techniques developed in the linear logic context – particularly those relying on biproducts – can be adapted to intuitionistic propositional logic. Instead of using a monad to encapsulate non-determinism or parallelism, we investigate how to reconstruct these effects using categorical structures that simulate the role of biproducts, even in their absence. In this way, we aim to propose a conceptually unified alternative for modelling such effects across both linear and non-linear logical frameworks.

Plan of the paper and contributions. In this paper, we introduce two lambda calculi for propositional logic – one modelling parallelism and the other algebraic combinations – based on minimal categorical structure inspired by biproducts, and avoiding the use of monads.

- In Section 2, we introduce the parallel lambda calculus $\lambda_{\parallel}$ for the full intuitionistic propositional logic. It is mostly based on the in-left-right-+-calculus (inlr-calculus for short) [12], with a slight modification. In the inlr-calculus the parallel of $\mathsf{inl}(t)$ and $\mathsf{inr}(u)$ reduces to a new term $\mathsf{inlr}(t,u)$. In our presentation we just keep $\mathsf{inl}(t) \parallel \mathsf{inr}(u)$, however, the two calculi are equivalent. This calculus follows the approach of [6] for the structure operator, where the parallel operator is neither idempotent, commutative, nor associative. Section 2.1 gives the syntax, deduction, and reduction rules. Section 2.2 restate the correctness properties of the calculus, proved in [12] and adapted to our setting in [10, Appendix A]. Section 2.3 introduces the category $\mathbf{Mag}_{\mathbf{Set}}$, which is the category whose objects are magmas and whose arrows are functions from the **Set** category. We also introduce the product $\oplus$ and prove several necessary properties. Section 2.4 provides the interpretation of $\lambda_{\parallel}$ within the category $\mathbf{Mag}_{\mathbf{Set}}$, and prove its soundness and adequacy results, that is, if one term reduces to another, they are interpreted by the same arrow, and if two closed proof-terms are interpreted by the same arrows, then they are computationally equivalent.
- In Section 3, we introduce the algebraic lambda calculus $\lambda^{\mathcal{S}}_{\mathsf{alg}}$ for the full intuitionistic propositional logic. This calculus is an extension of the $\lambda_{\parallel}$ calculus with scalars and follows the approach of the Algebraic lambda calculus [19].
  Section 3.1 gives the syntax, deduction, and reduction rules. Section 3.2 states the correctness properties of the calculus as straightforward adaptations of those from Section 2.2. Section 3.3 introduces the concept of action magmas and define the category

$$\frac{}{x : A, \Gamma \vdash x : A}\ \text{ax} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : A}{\Gamma \vdash t \parallel u : A}\ \text{par} \qquad \frac{}{\Gamma \vdash \star : \top}\ \top_i \qquad \frac{\Gamma \vdash t : \top \quad \Gamma \vdash u : C}{\Gamma \vdash \delta_\top(t,u) : C}\ \top_e$$

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \delta_\bot(t) : C}\ \bot_e \qquad \frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\ \Rightarrow_i \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t,u\rangle : A \wedge B}\ \wedge_i \qquad \frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \pi_1(t) : A}\ \wedge_{e1} \qquad \frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \pi_2(t) : B}\ \wedge_{e2}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{inl}(t) : A \vee B}\ \vee_{i1} \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \mathsf{inr}(t) : A \vee B}\ \vee_{i2}$$

$$\frac{\Gamma \vdash t : A \vee B \quad x : A, \Gamma \vdash u : C \quad y : B, \Gamma \vdash v : C}{\Gamma \vdash \delta_\vee(t, x.u, y.v) : C}\ \vee_e$$

Figure 1 The deduction rules of the $\lambda_\parallel$.

$\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$, whose objects are action magmas, and whose arrows are functions from the **Set** category. We also extend the product $\oplus$ into an action magma. Section 3.4 provides the interpretation of the algebraic lambda calculus within the category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$, and prove its soundness and adequacy.

- In Section 4, we present some concluding remarks.

2 The calculus $\lambda_\parallel$

2.1 Syntax, deduction, and reduction rules

The connectives in this logic are those found in standard intuitionistic propositional logic: $\top$, $\bot$, $\Rightarrow$, $\wedge$, and $\vee$. The syntax of the proof-terms are also standard, with the addition of the parallel construction as mentioned in the introduction.

$$\begin{array}{lll} t = & x \mid t \parallel t & \\ & \mid \star \mid \delta_\top(t,t) & (\top) \\ & \mid \delta_\bot(t) & (\bot) \\ & \mid \lambda x.t \mid t\ t & (\Rightarrow) \\ & \mid \langle t,t\rangle \mid \pi_1(t) \mid \pi_2(t) & (\wedge) \\ & \mid \mathsf{inl}(t) \mid \mathsf{inr}(t) \mid \delta_\vee(t, x.t, y.t) & (\vee) \end{array}$$

where x and y range over a countable set of variables.

The terms $\star$, $\lambda x.t$, $\langle t,u\rangle$, $\mathsf{inl}(t)$, and $\mathsf{inr}(t)$ are called introductions. The terms $\delta_\top(t,u)$, $\delta_\bot(t)$, $\pi_1(t)$, $\pi_2(t)$, and $\delta_\vee(t,x.u,y.v)$ are called eliminations. Variables and $t \parallel u$ are neither introductions nor eliminations, except for $\mathsf{inl}(t) \parallel \mathsf{inr}(u)$ which can be considered as an introduction. Free variables are defined as usual, and the substitution of x by u in t is written $(u/x)t$, with the usual renaming to avoid variable captures.

Our notation is inspired by logic. However, $\delta_\top(t,r)$ could also be written $t;r$ to denote sequencing, $\delta_\bot(t)$ as $\mathsf{error}(t)$ to denote raising an exception, $\delta_\vee(t,x.u,y.v)$ as $\mathsf{case}\ t\ \mathsf{of}\ \{\mathsf{inl}(x) \to u; \mathsf{inr}(y) \to v\}$ for pattern matching, and $\pi_1(t)$ and $\pi_2(t)$ as $\mathsf{fst}(t)$ and $\mathsf{snd}(t)$ for projections. From this viewpoint, the calculus can be seen as a programming language.

The also standard deduction rules are given in Figure 1.

The reduction relation is the contextual closure of the relation defined by the rules:

$$\delta_\top(\star, t) \longrightarrow t \quad (1)$$
$$(\lambda x.t)\ u \longrightarrow (u/x)t \quad (2)$$
$$\pi_1\langle t, u\rangle \longrightarrow t \quad (3)$$
$$\pi_2\langle t, u\rangle \longrightarrow u \quad (4)$$

$$\delta_\vee(\mathsf{inl}(t), x.v, y.w) \longrightarrow (t/x)v \quad (5)$$
$$\delta_\vee(\mathsf{inr}(u), x.v, y.w) \longrightarrow (u/y)w \quad (6)$$
$$\delta_\vee(\mathsf{inl}(t) \parallel \mathsf{inr}(u), x.v, y.w) \longrightarrow (t/x)v \parallel (u/y)w \quad (7)$$

$$\star \parallel \star \longrightarrow \star \quad (8)$$
$$(\lambda x.t) \parallel (\lambda x.u) \longrightarrow \lambda x.(t \parallel u) \quad (9)$$
$$\langle t_1, u_1\rangle \parallel \langle t_2, u_2\rangle \longrightarrow \langle t_1 \parallel t_2, u_1 \parallel u_2\rangle \quad (10)$$
$$\mathsf{inl}(t_1) \parallel \mathsf{inl}(t_2) \longrightarrow \mathsf{inl}(t_1 \parallel t_2) \quad (11)$$
$$\mathsf{inl}(t_1) \parallel (\mathsf{inl}(t_2) \parallel \mathsf{inr}(u_1)) \longrightarrow \mathsf{inl}(t_1 \parallel t_2) \parallel \mathsf{inr}(u_1) \quad (12)$$
$$\mathsf{inr}(u_1) \parallel \mathsf{inl}(t_1) \longrightarrow \mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1) \quad (13)$$
$$\mathsf{inr}(u_1) \parallel \mathsf{inr}(u_2) \longrightarrow \mathsf{inr}(u_1 \parallel u_2) \quad (14)$$
$$\mathsf{inr}(u_1) \parallel (\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_2)) \longrightarrow \mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1 \parallel u_2) \quad (15)$$
$$(\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1)) \parallel \mathsf{inl}(t_2) \longrightarrow \mathsf{inl}(t_1 \parallel t_2) \parallel \mathsf{inr}(u_1) \quad (16)$$
$$(\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1)) \parallel \mathsf{inr}(u_2) \longrightarrow \mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1 \parallel u_2) \quad (17)$$
$$(\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1)) \parallel (\mathsf{inl}(t_2) \parallel \mathsf{inr}(u_2)) \longrightarrow \mathsf{inl}(t_1 \parallel t_2) \parallel \mathsf{inr}(u_1 \parallel u_2) \quad (18)$$

The first group of reduction rules is standard, except for the last rule, which allows reducing inl and inr in parallel. The second group establishes the commutation of the parallel construction with all the connectives. The parallel construction commutes with introductions, as seen in the first three rules of the second group for the cases of $\top$, $\Rightarrow$, and $\wedge$. The case of $\vee$ is left to the next eight rules since multiple cases must be considered due to the fact that there are two normal forms for disjunctions, and disjunction is neither commutative nor associative in Natural Deduction.

Instead of commuting the parallel construction with the introduction of disjunction, we could have considered doing so with its elimination, using the rule $\delta_\vee(t \parallel u, x.v, y.w) \longrightarrow \delta_\vee(t, x.u, y.w) \parallel \delta_\vee(u, x.v, y.w)$, which can also be proven valid in our model (the proof of soundness for this rule is given in [10, Appendix C]). However, we chose to consider commutation with the introduction, as it leads to a better introduction property: a closed irreducible proof of $\top$ is $\star$, a closed irreducible proof of an implication is a lambda abstraction, a closed irreducible proof of a conjunction is a pair, and a closed irreducible proof of a disjunction is either inl, inr, or the parallel of both (see Theorem 2.4).

Along this paper if R is a relation, we write R^* for its transitive-reflexive closure.

2.2 Correctness

The correctness properties of this section have been proven in a preprint for the inlr calculus [12], and the proofs are the same for the $\lambda_\parallel$ calculus.

▶ **Theorem 2.1** (Subject reduction [12, Appendix A.1]). *If $\Gamma \vdash t : A$ and $t \longrightarrow u$, then $\Gamma \vdash u : A$.*

▶ **Theorem 2.2** (Strong normalisation [12, Appendix A.4]). *If $\Gamma \vdash t : A$ then t is strongly normalising.*

▶ **Theorem 2.3** (Confluence). *The $\lambda_\parallel$ calculus is confluent.*

Proof. The relation $\longrightarrow$ from Section 2.1 is left linear and has no critical pairs. And, as proved in [17, Theorem 6.8], higher-order left linear systems without critical pairs are confluent. ◀

▶ **Theorem 2.4** (Introduction property [12, Appendix A.3])**.** *Let $\vdash t : A$ be irreducible. Then: If $A = \top$, $t = \star$. A is not $\bot$. If $A = B \Rightarrow C$, $t = \lambda x.u$. If $A = B \wedge C$, $t = \langle u, v\rangle$. If $A = B \vee C$, $t = \mathsf{inl}(u)$, $t = \mathsf{inr}(v)$, or $t = \mathsf{inl}(u) \parallel \mathsf{inr}(v)$.*

2.3 The category $\mathrm{Mag}_{\mathbf{Set}}$

▶ **Definition 2.5** (The category $\mathbf{Mag_{Set}}$)**.** *The category $\mathbf{Mag_{Set}}$ is determined by the following data: Objects are magmas $(A, *_A)$. Arrows are maps in $\mathbf{Set}$.*

▶ Remark 2.6. Note that $\mathbf{Mag_{Set}}$ is not an algebraic category, in the sense that the natural arrows for the algebraic structure (such as homomorphisms) are not considered. Instead, the objects are plain magmas, without any algebraic property, and the arrows are simply functions in $\mathbf{Set}$, with composition and identities inherited from $\mathbf{Set}$. The reasons for this choice are discussed in detail in Section 4.

The next definitions give some objects and arrows used later.

▶ **Definition 2.7.** *Let A and B be objects in $\mathbf{Mag_{Set}}$, we define the following objects.*

- *$\emptyset$ with the empty map as its operation.*
- *$\{\star\}$ with the operation defined as $\star *_{\{\star\}} \star = \star$.*
- *$A \times B$ with the operation defined as $(a_1, b_1) *_{A\times B} (a_2, b_2) = (a_1 *_A a_2, b_1 *_B b_2)$.*
- *$[A \to B]$ with the operation defined as $(f *_{[A\to B]} g)(a) = f(a) *_B g(a)$.*

▶ **Theorem 2.8** (CCC)**.** *The category $\mathbf{Mag_{Set}}$ is Cartesian closed.*

Proof. We need to prove that there is an adjunction $A \times _ \dashv [A \to _]$ in $\mathbf{Mag_{Set}}$. For all A, B objects in $\mathbf{Mag_{Set}}$, we have $Hom_{\mathbf{Mag_{Set}}}(A, B) = Hom_{\mathbf{Set}}(A, B)$. Since, $A \times B$ and $[B \to C]$ are objects in $\mathbf{Mag_{Set}}$, by the adjunction in $\mathbf{Set}$, we have $Hom_{\mathbf{Mag_{Set}}}(A \times B, C) = Hom_{\mathbf{Set}}(A \times B, C) \simeq Hom_{\mathbf{Set}}(A, [B \to C]) = Hom_{\mathbf{Mag_{Set}}}(A, [B \to C])$. ◀

As usual, the disjoint union of two sets A and B is denoted by $A \uplus B$ and is defined as $A \uplus B = (A \times \{0\}) \cup (B \times \{1\})$ where 0 and 1 are distinct elements that do not belong to either to A or B. If we want $A \uplus B$ to be an object in $\mathbf{Mag_{Set}}$, we must define its magma operation. For example, we can choose $(a_1, 0) *_\uplus (a_2, 0) = (a_1 *_A a_2, 0)$, and similarly $(b_1, 1) *_\uplus (b_2, 1) = (b_1 *_B b_2, 1)$. However, it is not clear how to define $(a, 0) *_\uplus (b, 1)$. We could choose to define it as either $(a, 0)$ or $(b, 1)$, but this would break the symmetry. This is the same problem encountered in $\mathsf{inl}(t) \parallel \mathsf{inr}(u)$. Indeed, in the sequent $\vdash \mathsf{inl}(t) \parallel \mathsf{inr}(u) : A \vee B$, we are given both a proof of A and a proof of B. However, to produce a proof of $A \vee B$, we would typically need to discard one of them, whereas with the parallel operator, we can retain both. This is why we need to define a new operation, which we call pokeball product and denote by $\oplus\!\!\!\!\bigcirc$.

▶ **Definition 2.9** (The pokeball product)**.** *We define $A \oplus\!\!\!\!\bigcirc\, B = (A \uplus B) \cup (A \times B)$.*

▶ **Definition 2.10** (The operation $*_{\oplus}$). *Let $(A, *_A), (B, *_B) \in \mathsf{Obj}(\mathbf{Mag_{Set}})$. We define $*_{A\oplus B}$ ($*_{\oplus}$ for short) as $(A \oplus B) \times (A \oplus B) \xrightarrow{*_{\oplus}} A \oplus B$ given by*

$$(c_1, c_2) \mapsto \begin{cases} (a_1 *_A a_2, 0) & \text{if } c_1 = (a_1, 0) \text{ and } c_2 = (a_2, 0) \\ (a, b) & \text{if } c_1 = (a, 0) \text{ and } c_2 = (b, 1) \\ (a_1 *_A a_2, b) & \text{if } c_1 = (a_1, 0) \text{ and } c_2 = (a_2, b) \\ (a, b) & \text{if } c_1 = (b, 1) \text{ and } c_2 = (a, 0) \\ (b_1 *_B b_2, 1) & \text{if } c_1 = (b_1, 1) \text{ and } c_2 = (b_2, 1) \\ (a, b_1 *_B b_2) & \text{if } c_1 = (b_1, 1) \text{ and } c_2 = (a, b_2) \\ (a_1 *_A a_2, b) & \text{if } c_1 = (a_1, b) \text{ and } c_2 = (a_2, 0) \\ (a, b_1 *_B b_2) & \text{if } c_1 = (a, b_1) \text{ and } c_2 = (b_2, 1) \\ (a_1 *_A a_2, b_1 *_B b_2) & \text{if } c_1 = (a_1, b_1) \text{ and } c_2 = (a_2, b_2) \end{cases}$$

▶ **Definition 2.11.** *Let $A \xrightarrow{f} A'$ and $B \xrightarrow{g} B'$. The arrow $A \oplus B \xrightarrow{f\oplus g} A' \oplus B'$ is defined as follows.*

$$c \mapsto \begin{cases} (f(a), 0) & \text{if } c = (a, 0) \\ (g(b), 1) & \text{if } c = (b, 1) \\ (f(a), g(b)) & \text{if } c = (a, b) \end{cases}$$

The pokeball product entails a bifunctorial functor.

▶ **Lemma 2.12** (Bifunctoriality). *Let $A \xrightarrow{f} A'$, $A \xrightarrow{f'} A'$, $B \xrightarrow{g} B'$, and $B \xrightarrow{g'} B'$. Then $A \oplus _$ is a functor, and it is bifunctorial, that is $(f \oplus g) \circ (f' \oplus g') = (f \circ f') \oplus (g \circ g')$.*

Proof.

- We prove that $A \oplus _$ is a functor. That is, we need to prove the commutation of the following diagrams.

$$\begin{array}{ccc} A \oplus B & \xrightarrow{\mathsf{id}\oplus g} & A \oplus B' \\ & \searrow^{\mathsf{id}\oplus(h\circ g)} & \downarrow{\scriptstyle \mathsf{id}\oplus h} \\ & & A \oplus B'' \end{array} \qquad\qquad A \oplus B \xrightarrow[\mathsf{id}_{A\oplus B}]{\mathsf{id}_A \oplus \mathsf{id}_B} A \oplus B$$

We prove the diagrams by giving their evaluations.

$$\begin{array}{ccc} (a,0) & \xmapsto{\mathsf{id}\oplus g} & (a,0) \\ & \searrow^{\mathsf{id}\oplus(h\circ g)} & \downarrow{\scriptstyle \mathsf{id}\oplus h} \\ & & (a,0) \end{array} \qquad \begin{array}{ccc} (b,1) & \xmapsto{\mathsf{id}\oplus g} & (g(b),1) \\ & \searrow^{\mathsf{id}\oplus(h\circ g)} & \downarrow{\scriptstyle \mathsf{id}\oplus h} \\ & & (h(g(b)),1) \end{array} \qquad \begin{array}{ccc} (a,b) & \xmapsto{\mathsf{id}\oplus g} & (a,g(b)) \\ & \searrow^{\mathsf{id}\oplus(h\circ g)} & \downarrow{\scriptstyle \mathsf{id}\oplus h} \\ & & (a,h(g(b))) \end{array}$$

$$(a,0) \xmapsto[\mathsf{id}_{A\oplus B}]{\mathsf{id}_A \oplus \mathsf{id}_B} (a,0) \qquad (b,1) \xmapsto[\mathsf{id}_{A\oplus B}]{\mathsf{id}_A \oplus \mathsf{id}_B} (b,1) \qquad (a,b) \xmapsto[\mathsf{id}_{A\oplus B}]{\mathsf{id}_A \oplus \mathsf{id}_B} (a,b)$$

The proof of $_ \oplus B$ is also analogous.

- To prove that it is bifunctorial it suffices to prove that the following diagram commutes [15, Proposition 3.1].

$$\begin{array}{ccc} A \oplus B & \xrightarrow{\mathsf{id}\oplus g} & A \oplus B' \\ {\scriptstyle f\oplus\mathsf{id}}\downarrow & & \downarrow{\scriptstyle f\oplus\mathsf{id}} \\ A' \oplus B & \xrightarrow[\mathsf{id}\oplus g]{} & A' \oplus B' \end{array}$$

We prove this diagram by giving its possible evaluations.

$$\begin{array}{ccc} (a,0) & \xmapsto{\mathsf{id}\oplus g} & (a,0) \\ {\scriptstyle f\oplus\mathsf{id}}\downarrow & & \downarrow{\scriptstyle f\oplus\mathsf{id}} \\ (f(a),0) & \xmapsto[\mathsf{id}\oplus g]{} & (f(a),0) \end{array} \qquad \begin{array}{ccc} (b,1) & \xmapsto{\mathsf{id}\oplus g} & (g(b),1) \\ {\scriptstyle f\oplus\mathsf{id}}\downarrow & & \downarrow{\scriptstyle f\oplus\mathsf{id}} \\ (b,1) & \xmapsto[\mathsf{id}\oplus g]{} & (g(b),1) \end{array} \qquad \begin{array}{ccc} (a,b) & \xmapsto{\mathsf{id}\oplus g} & (a,g(b)) \\ {\scriptstyle f\oplus\mathsf{id}}\downarrow & & \downarrow{\scriptstyle f\oplus\mathsf{id}} \\ (f(a),b) & \xmapsto[\mathsf{id}\oplus g]{} & (f(a),g(b)) \end{array}$$

◀

While the pokeball product will not be a coproduct in $\mathbf{Mag}_{\mathbf{Set}}$ (see Remark 2.17), we can define some maps that will allow us to interpret the disjunction with the pokeball product in a manner similar to a coproduct. First, we note that the usual injections i_1 and i_2 are magma homomorphisms.

▶ **Lemma 2.13.** *The maps $A \xrightarrow{i_1} A \oplus B$ and $B \xrightarrow{i_2} A \oplus B$ are homomorphisms.*

Proof.

- $i_1(a_1) *_\oplus i_1(a_2) = (a_1,0) *_\oplus (a_2,0) = (a_1 *_A a_2, 0) = i_1(a_1 *_A a_2)$
- $i_2(b_1) *_\oplus i_2(b_2) = (b_1,1) *_\oplus (b_2,1) = (b_1 *_B b_2, 1) = i_2(b_1 *_B b_2)$

◀

We can define the inclusion $A \times B \xrightarrow{\text{inc}} A \oplus B$ by $(a,b) \mapsto (a,b)$. An interesting property is that after two injections and the operation $*_\oplus$, we can recover the original element, as stated by the following lemma.

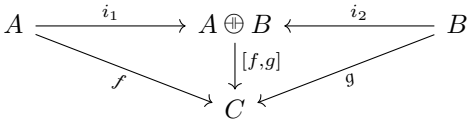
▶ **Lemma 2.14.** $*_\oplus \circ (i_1 \times i_2) = \text{inc}$.

Proof. Let $a \in A$ and $b \in B$. Then $*_\oplus \circ (i_1 \times i_2)(a,b) = (a,0) *_\oplus (b,1) = (a,b)$. ◀

▶ **Definition 2.15.** *Let $A \xrightarrow{f} C$ and $B \xrightarrow{g} C$ be arrows in $\mathbf{Mag}_{\mathbf{Set}}$. We define the arrow $A \oplus B \xrightarrow{[f,g]} C$ as follows.*

$$c \mapsto \begin{cases} f(a) & \text{if } c = (a,0) \\ g(b) & \text{if } c = (b,1) \\ f(a) *_C g(b) & \text{if } c = (a,b) \end{cases}$$

▶ **Lemma 2.16** (Weak coproduct)**.** *Then map $[f,g]$ makes the following diagram commute*



Proof.

- Left triangle: $[f,g](i_1(a)) = [f,g](a,0) = f(a)$.
- Right triangle: $[f,g](i_2(b)) = [f,g](b,1) = g(b)$.

◀

▶ Remark 2.17 (Not a coproduct). Lemma 2.16 does not imply that $A \oplus B$ is a coproduct in $\mathbf{Mag}_{\mathbf{Set}}$, since the map $[f,g]$ is clearly not unique.

However, in the category **Mag** of magmas, whose arrows are homomorphism (which is not Cartesian closed, and so it is not a good candidate for us), the map $[f,g]$ is indeed a coproduct, since it is easy to prove that it is an homomorphism, and its unicity can be proven using the homomorphism property as follows.

Let $h : A \oplus B \to C \in \mathsf{Arr}(\mathbf{Mag})$ such that $h(i_1(a)) = h(a,0) = f(a)$ and $h(i_2(b)) = h(b,1) = g(b)$. Then, $h(a,b) = h((a,0) *_\oplus (b,1)) = h(a,0) *_C h(b,1) = f(a) *_C g(b)$. Therefore, if arrows are homomorphisms, $h = [f,g]$.

▶ **Definition 2.18.** *Let A and B be objects in $\mathbf{Mag_{Set}}$, we define the following arrows.*

- $A \xrightarrow{!} \{\star\}$ *defined by* $a \mapsto \star$.
- $A \xrightarrow{\Delta} A \times A$ *defined by* $a \mapsto (a, a)$.
- $(A \oplus B) \times C \xrightarrow{d} (A \times C) \oplus (B \times C)$ *defined by*

$$(e, c) \mapsto \begin{cases} ((a, c), 0) & \text{if } e = (a, 0) \\ ((b, c), 1) & \text{if } e = (b, 1) \\ ((a, c), (b, c)) & \text{if } e = (a, b) \end{cases}$$

- $A \times A \xrightarrow{*} A$ *defined by* $(a_1, a_2) \mapsto a_1 *_A a_2$.
- $\delta = (\mathsf{id} \times \sigma \times \mathsf{id}) \circ (\mathsf{id} \times \Delta)$. *That is,* $A \times B \times C \xrightarrow{\delta} A \times C \times B \times C$.

2.4 Model of the $\lambda_{\|}$ calculus

2.4.1 Interpretation

The interpretation of the propositions and contexts of the proof system is the following:

Interpretation of propositions			Interpretation of contexts		
$[\![\top]\!]$	=	$\{\star\}$	$[\![\emptyset]\!]$	=	$\{\star\}$
$[\![\bot]\!]$	=	$\emptyset$	$[\![x : A, \Gamma]\!]$	=	$[\![A]\!] \times [\![\Gamma]\!]$
$[\![A \Rightarrow B]\!]$	=	$[[\![A]\!] \to [\![B]\!]]$			
$[\![A \wedge B]\!]$	=	$[\![A]\!] \times [\![B]\!]$			
$[\![A \vee B]\!]$	=	$[\![A]\!] \oplus [\![B]\!]$			

The interpretation of the deduction rules is the following We refer to each deduction rule by its last sequent, since the system is syntax directed.

$$[\![x : A, \Gamma \vdash x : A]\!] = [\![A]\!] \times [\![\Gamma]\!] \xrightarrow{\pi_1} [\![A]\!]$$

$$[\![\Gamma \vdash t \parallel u : A]\!] = [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t \times u} [\![A]\!] \times [\![A]\!] \xrightarrow{*_{[\![A]\!]}} [\![A]\!]$$

$$[\![\Gamma \vdash \star : \top]\!] = [\![\Gamma]\!] \xrightarrow{!} \{\star\} = [\![\top]\!]$$

$$[\![\Gamma \vdash \delta_\top(t, u) : C]\!] = [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t \times u} \{\star\} \times [\![C]\!] \xrightarrow{\pi_2} [\![C]\!]$$

$$[\![\Gamma \vdash \delta_\bot(t) : C]\!] = [\![\Gamma]\!] \xrightarrow{t} \emptyset \xrightarrow{\emptyset} [\![C]\!]$$

$$[\![\Gamma \vdash \lambda x.t : A \Rightarrow B]\!] = [\![\Gamma]\!] \xrightarrow{\eta^{[\![A]\!]}} [[\![A]\!] \to [\![A]\!] \times [\![\Gamma]\!]] \xrightarrow{[[\![A]\!] \to t]} [[\![A]\!] \to [\![B]\!]]$$

$$[\![\Gamma \vdash t\ u : B]\!] = [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t \times u} [[\![A]\!] \to [\![B]\!]] \times [\![A]\!] \xrightarrow{\varepsilon} [\![B]\!]$$

$$[\![\Gamma \vdash \langle t, u \rangle : A \wedge B]\!] = [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t \times u} [\![A]\!] \times [\![B]\!]$$

$$[\![\Gamma \vdash \pi_1(t) : A]\!] = [\![\Gamma]\!] \xrightarrow{t} [\![A \wedge B]\!] = [\![A]\!] \times [\![B]\!] \xrightarrow{\pi_1} [\![A]\!]$$

$$[\![\Gamma \vdash \pi_2(t) : A]\!] = [\![\Gamma]\!] \xrightarrow{t} [\![A \wedge B]\!] = [\![A]\!] \times [\![B]\!] \xrightarrow{\pi_2} [\![B]\!]$$

$$[\![\Gamma \vdash \mathsf{inl}(t) : A \vee B]\!] = [\![\Gamma]\!] \xrightarrow{t} [\![A]\!] \xrightarrow{i_1} [\![A]\!] \oplus [\![B]\!]$$

$$[\![\Gamma \vdash \mathsf{inr}(t) : A \vee B]\!] = [\![\Gamma]\!] \xrightarrow{t} [\![B]\!] \xrightarrow{i_2} [\![A]\!] \oplus [\![B]\!]$$

$$[\![\Gamma \vdash \delta_\vee(t, x.u, y.v) : C]\!] = [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t \times \mathsf{id}} ([\![A]\!] \oplus [\![B]\!]) \times [\![\Gamma]\!] \xrightarrow{d} ([\![A]\!] \times [\![\Gamma]\!]) \oplus ([\![B]\!] \times [\![\Gamma]\!]) \xrightarrow{[u,v]} [\![C]\!]$$

In some diagrams we may write A instead of $[\![A]\!]$, to simplify the notation.

2.4.2 Soundness

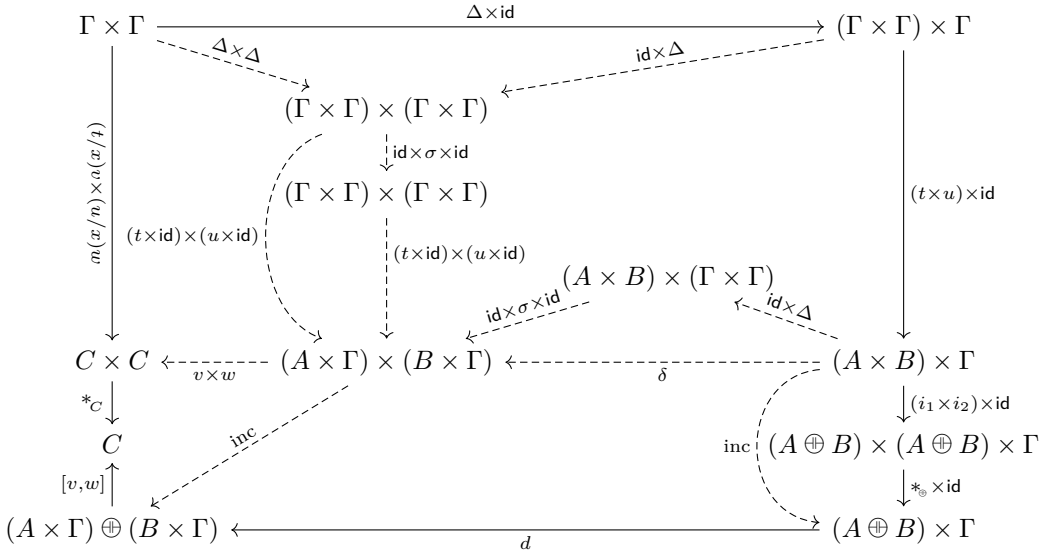
The soundness of the model is proven in Theorem 2.20. As usual, we require the following substitution lemma. Its proof is done by induction on t and given in [10, Appendix B.1].

▶ **Lemma 2.19** (Substitution). *If* $x : A, \Gamma \vdash t : B$ *and* $\Gamma \vdash u : A$*, then* $[\![\Gamma \vdash (u/x)t : B]\!] = [\![\Gamma \vdash t : B]\!] \circ ([\![\Gamma \vdash u : A]\!] \times \mathsf{id}) \circ \Delta$.

▶ **Theorem 2.20** (Soundness). *If* $t \longrightarrow r$ *and* $\Gamma \vdash t : A$*, then* $[\![\Gamma \vdash t : A]\!] = [\![\Gamma \vdash r : A]\!]$.

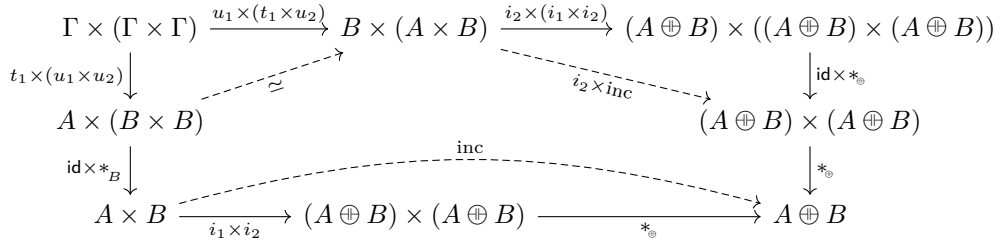
Proof. By induction on the relation $\longrightarrow$. The full proof is given in [10, Appendix B.2]. We only show two interesting cases here, involving the parallel operator.

- Rule $\delta_\vee(\mathsf{inl}(t) \parallel \mathsf{inr}(u), x.v, y.w) \longrightarrow (t/x)v \parallel (u/y)w$. The commuting diagram is



  The top diagram commutes by the functoriality of $\times$. The upper-left diagram commutes by Lemma 2.19 and the functoriality of $\times$. The upper-middle diagram commutes by coherence, when precomposed with Δs. The upper-right-top diagram commutes by the naturality of $(\mathsf{id} \times \sigma \times \mathsf{id}) \circ (\mathsf{id} \times \Delta)$. The upper-right-bottom diagram is the definition of the map δ. The bottom-left diagram commutes by definition of $[v, w]$. The bottom-right diagram commutes by definition of $*_\oplus$. The bottom-middle diagram commutes since δ is a restriction of d.

- Rule $\mathsf{inr}(u_1) \parallel (\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_2)) \longrightarrow \mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1 \parallel u_2)$. The commuting diagram is



  The upper-left diagram commutes by coherence. The upper-right diagram commutes by the functoriality of $\times$ and Lemma 2.14. The bottom diagram commutes by Lemma 2.14. The middle diagram commutes since $*_\oplus \circ (i_2 \times \mathrm{inc})(b_1, (a, b_2)) = (b_1, 1) *_\oplus (a, b_2) = (a, b_1 *_B b_2)$. ◀

▶ Remark 2.21.

- The case of the rule $\delta_\vee(\mathsf{inl}(t) \parallel \mathsf{inr}(u), x.v, y.w) \longrightarrow (t/x)v \parallel (u/y)w$ uses the fact that δ is a restriction of d. Indeed, $A \oplus B$ includes $A \times B$, and thus, $\mathsf{inl}(t) \parallel \mathsf{inr}(u)$ can be seen as a pair of terms.
- The case of the rule $\mathsf{inr}(u_1) \parallel (\mathsf{inl}(t_1) \parallel \mathsf{inr}(u_2)) \longrightarrow \mathsf{inl}(t_1) \parallel \mathsf{inr}(u_1 \parallel u_2)$ is solved by the definition of $*_\oplus$. Indeed, we do not ask all the operations to be commutative or associative, but we can still use an operation with these properties to solve the problem.

2.4.3 Adequacy

A proof of completeness would mean that if two closed proofs have the same interpretation, they both reduce to the same irreducible proof. However, this is too much to ask for in a calculus without η-reduction. Indeed, $\vdash \lambda x.tx : A \Rightarrow B$ and $\vdash t : A \Rightarrow B$ have the same interpretation, but they do not reduce to the same proof. We can only ask for the following weaker property, which is usually called adequacy. That is, if two proof-terms have the same interpretation, then they are computationally equivalent. The notion of computational equivalence is that no other program can distinguish between them. That is, putting them in any context will yield the same result. It is sufficient, however, to consider only elimination contexts of some proposition large enough to distinguish between two programs, such as $\top \vee \top$, the usual booleans.

Indeed, if K is one of these contexts, putting two terms t_1 and t_2 inside K is sufficient, and testing whether an introduction term such as $\langle K, u\rangle$ behaves the same when t_1 or t_2 are put inside K does not add any additional information. Only K, the elimination context, is sufficient to distinguish between the two terms.

The notion of an elimination context is given below. Without loss of generality, we consider the elimination context to always produce proof-terms for smaller propositions.

▶ **Definition 2.22** (Elimination context). *An elimination context is a proof-term produced by the following grammar, where $[\cdot]$ denotes a distinguished variable.*

$$K := [\cdot] \mid K\ t \mid \pi_1(K) \mid \pi_2(K) \mid \delta_\vee(K, x.t, y.u)$$

where in the proof $\delta_\vee(K, x.t, y.u)$, if $[\cdot] : C \vdash K : A \vee B$, then $x : A \vdash t : D$ and $y : B \vdash u : D$, with $|D| < |A \vee B|$. As usual, we write $K[t]$ for $(t/[\cdot])K$.

We define the computational equivalence as follows.

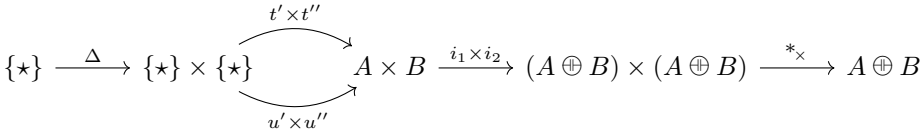
▶ **Definition 2.23** (Computational equivalence). *Two proofs $\vdash t : A$ and $\vdash u : A$ are computationally equivalent, written $t \sim u$, if for each elimination context $[\cdot] : A \vdash K : \top \vee \top$, there exists an irreducible proof $\vdash v : \top \vee \top$ such that $K[t] \longrightarrow^* v$ and $K[u] \longrightarrow^* v$.*

Finally, we can state and prove the adequacy of the model as follows.

▶ **Theorem 2.24** (Adequacy). *If $[\![\vdash t : A]\!] = [\![\vdash u : A]\!]$, then $t \sim u$.*

Proof. By induction on the structure of A. The full details are given in [10, Appendix B.3]. We only show one interesting case here. If $A = B \vee C$, then, by Theorems 2.1, 2.2, and 2.4, $t \longrightarrow^* \mathsf{inl}(t')$, $t \longrightarrow^* \mathsf{inr}(t')$, or $t \longrightarrow^* \mathsf{inl}(t') \parallel \mathsf{inr}(t'')$; and $u \longrightarrow^* \mathsf{inl}(u')$, $u \longrightarrow^* \mathsf{inr}(u')$, $u \longrightarrow^* \mathsf{inl}(u') \parallel \mathsf{inr}(u'')$.

However, by Theorem 2.20, and the fact that $[\![\vdash t : A]\!] = [\![\vdash u : A]\!]$, we have that if $t \longrightarrow^* \mathsf{inl}(t')$ then $u \longrightarrow^* \mathsf{inl}(u')$, if $t \longrightarrow^* \mathsf{inr}(t')$ then $u \longrightarrow^* \mathsf{inr}(u')$, and if $t \longrightarrow^* \mathsf{inl}(t') \parallel \mathsf{inr}(t'')$ then $u \longrightarrow^* \mathsf{inl}(u') \parallel \mathsf{inr}(u'')$. We must consider these three cases. We show here only the third case as example. We have



Thus, $[\![\vdash t' : A]\!] = [\![\vdash u' : A]\!]$ and $[\![\vdash t'' : B]\!] = [\![\vdash u'' : B]\!]$, and so, by the induction hypothesis $t' \sim u'$ and $t'' \sim u''$. Hence, $t \sim u$. ◀

3 The calculus $\lambda^{\mathcal{S}}_{\mathsf{alg}}$

3.1 Syntax, deduction, and reduction rules

In this section, we consider the $\lambda^{\mathcal{S}}_{\mathsf{alg}}$ calculus as an extension with scalars of the $\lambda_{\|}$ calculus. This calculus is similar to the Algebraic lambda calculus [19], but considering all the logical connectives, making it closer to the $\odot^{\mathcal{S}}$-calculus [8]. The idea is to consider linear combinations of terms, as in $\mathfrak{a} \cdot t \parallel \mathfrak{b} \cdot u$, where we continue writing $\|$ for the operation instead of the more common $+$ from algebraic calculi, to emphasise its connection with the language defined in the previous section.

In the Algebraic lambda calculus [19], the scalars belong to a semiring with zero and unity (i.e. a rig) and it is claimed not to use a ring since there are some confluence problems with negative scalars in its untyped setting. Indeed, a term $Y_t = (\lambda x.xx \parallel t)(\lambda x.xx \parallel t)$ reducing to $Y_t \parallel t$ is the source of non-confluence when considering $\mathfrak{a} \cdot Y_t \parallel -\mathfrak{a} \cdot Y_t$, which may reduce to $t \parallel \cdots \parallel t$ for any amount of t in parallel (even zero), since at any point $\mathfrak{a} \cdot Y_t$ may be cancelled with $-\mathfrak{a} \cdot Y_t$ on these calculi. There are several solutions to this problem. One is to consider only positive scalars, as done in the Algebraic lambda calculus. Another one is to consider a call-by-value beta reduction together with a linearity rule, as done in [1] (specifically, the solution involves restricting some rewriting rules to closed normal forms, which forces a call-by-value strategy). However, the simpler solution to this lack of confluence is to consider a typed setting ensuring strong normalisation, where the term Y_t is not typable. For an extensive discussion on the different design choices for algebraic calculi, see [2]. In our case, we relax all the restrictions on scalars and just require them to belong to a bi-magma, as defined next.

▶ **Definition 3.1** (Bi-magma). *A bi-magma is a set with two operations* $(\mathcal{S}, +, \cdot)$.

The syntax of the $\lambda^{\mathcal{S}}_{\mathsf{alg}}$ calculus extends the $\lambda_{\|}$ calculus introduced in Section 2.1 by incorporating scalars from a fixed bi-magma $\mathcal{S}$, as follows: The term $\star$ is replaced with $\star_{\mathfrak{s}}$, and a new term $\mathfrak{s} \bullet t$ is added, where $\mathfrak{s} \in \mathcal{S}$ and t is a term.

The deduction rules are also those given in Section 2.1, with the exception of rule $\top_i$, and the addition of new families of rules "$\top_i(\mathfrak{s})$" and "prod($\mathfrak{s}$)" as follows.

$$\frac{}{\Gamma \vdash \star_{\mathfrak{s}} : \top}\ \top_i(\mathfrak{s}) \qquad \frac{\vdash t : A}{\Gamma \vdash \mathfrak{s} \bullet t : A}\ \text{prod}(\mathfrak{s})$$

As with the rule par, the rules prod($\mathfrak{s}$) do not introduce new provable formulas. They can be read as "If we have a proof of A, then we have a proof of A".

The reduction relation is similar to that of the $\lambda_{\|}$ calculus given in Section 2.1, with modifications to the elimination of $\top$ and the commutation of $\|$ with $\top_i(\mathfrak{s})$, since $\top_i$ now includes scalars:

$$\delta_{\top}(\star_{\mathfrak{s}}, t) \longrightarrow \mathfrak{s} \bullet t \qquad \star_{\mathfrak{s}_1} \parallel \star_{\mathfrak{s}_2} \longrightarrow \star_{\mathfrak{s}_1 + \mathfrak{s}_2}$$

Additionally, a third group is added for the commutation rules of prod($\mathfrak{s}$) with all the connectives. Indeed, just as with the rule par, the rule prod($\mathfrak{s}$) can appear between the introduction and the elimination of a connective. As in the case of the par rule, we must commute it with either the elimination or the introduction. Following the same convention, we always commute the prod($\mathfrak{s}$) rule with introductions:

$$\mathfrak{s}_1 \bullet \star_{\mathfrak{s}_2} \longrightarrow \star_{\mathfrak{s}_1 \cdot \mathfrak{s}_2} \qquad \mathfrak{s} \bullet \mathsf{inl}(t) \longrightarrow \mathsf{inl}(\mathfrak{s} \bullet t)$$
$$\mathfrak{s} \bullet \lambda x.t \longrightarrow \lambda x.\mathfrak{s} \bullet t \qquad \mathfrak{s} \bullet \mathsf{inr}(t) \longrightarrow \mathsf{inr}(\mathfrak{s} \bullet t)$$
$$\mathfrak{s} \bullet \langle t, v\rangle \longrightarrow \langle \mathfrak{s} \bullet t, \mathfrak{s} \bullet v\rangle \qquad \mathfrak{s} \bullet (\mathsf{inl}(t) \parallel \mathsf{inr}(v)) \longrightarrow \mathsf{inl}(\mathfrak{s} \bullet t) \parallel \mathsf{inr}(\mathfrak{s} \bullet v)$$

3.2 Correctness

The correctness properties are straightforward adaptations of the proofs of Theorems 2.1, 2.2, 2.3, and 2.4, and are therefore omitted. The only significant difference is that the introduction property now states that if $\vdash t : \top$ is irreducible, then $t = \star_{\mathfrak{s}}$ for some $\mathfrak{s} \in \mathcal{S}$.

3.3 The category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$

▶ **Definition 3.2** (Action magma). *An action magma $(A, *_A, \bullet_A)$ over $\mathcal{S}$ consists of a magma $(A, *_A)$ and a bi-magma $(\mathcal{S}, +, \cdot)$, with a map $\mathcal{S} \times A \xrightarrow{\bullet_A} A$ such that it distributes with respect to $*_A$, that is $\mathfrak{s} \bullet_A (a_1 *_A a_2) = \mathfrak{s} \bullet_A a_1 *_A \mathfrak{s} \bullet_A a_2$.*

▶ **Definition 3.3** (The category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$). *The category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$ is determined by the following data: Objects are action magmas over the fixed bi-magma $\mathcal{S}$. Arrows are maps in $\mathbf{Set}$.*

▶ **Theorem 3.4** (CCC). *The category $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$ is Cartesian closed.*

Proof. The argument is the same as the one given in Theorem 2.8. ◀

We now extend the pokeball to the Action Magmas by defining its action.

▶ **Definition 3.5** (The operation $\bullet_\oplus$). *Let $(A, *_A, \bullet_A), (B, *_B, \bullet_B) \in \mathsf{Obj}(\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}})$. We define $\bullet_{A\oplus B}$ ($\bullet_\oplus$ for short) as $\mathcal{S} \times (A \oplus B) \xrightarrow{\bullet_\oplus} A \oplus B$ given by*

$$(\mathfrak{s}, c) \mapsto \begin{cases} (\mathfrak{s} \bullet_A a, 0) & \text{if } c = (a, 0) \\ (\mathfrak{s} \bullet_B b, 1) & \text{if } c = (b, 1) \\ (\mathfrak{s} \bullet_A a, \mathfrak{s} \bullet_B b) & \text{if } c = (a, b) \end{cases}$$

▶ **Definition 3.6.** *Let $A, B \in \mathsf{Obj}(\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}})$. The following are also objects in $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$.*

- *$(A \oplus B, *_\oplus, \bullet_\oplus)$ with the action given in Definition 3.5.*
- *$(\emptyset, *_\emptyset, \bullet_\emptyset)$ with the action defined as the empty map.*
- *$(\{\star\}, *_{\{\star\}}, \bullet_{\{\star\}})$ with the action defined as $\mathfrak{s} \bullet_{\{\star\}} \star = \star$.*
- *A bi-magma $(\mathcal{S}, +, \cdot)$ with the magma operation defined by $+$ and the action defined by $\cdot$.*
- *$(A \times B, *_{A\times B}, \bullet_{A\times B})$ with the action defined by $\mathfrak{s} \bullet_{A\times B} (a, b) = (\mathfrak{s} \bullet_A a, \mathfrak{s} \bullet_B b)$.*
- *$([A \to B], *_{[A\to B]}, \bullet_{[A\to B]})$ with the action defined by $(\mathfrak{s} \bullet_{[A\to B]} f)(a) = \mathfrak{s} \bullet_B f(a)$.*

▶ **Definition 3.7.** *Let A be an object in $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$, we define the following arrows.*

- *$\{\star\} \xrightarrow{\mathfrak{s}} \mathcal{S}$ is defined as $\star \mapsto \mathfrak{s}$.*
- *$\mathcal{S} \times A \xrightarrow{\bullet_A} A$ is defined as $(\mathfrak{s}, a) \mapsto \mathfrak{s} \bullet_A a$.*
- *$A \xrightarrow{\hat{\mathfrak{s}}} A$ is defined as $A \xrightarrow{\rho} \{\star\} \times A \xrightarrow{\mathfrak{s}\times\mathsf{id}} \mathcal{S} \times A \xrightarrow{\bullet_A} A$.*

3.4 Model of the $\lambda^{\mathcal{S}}_{\mathsf{alg}}$ calculus

3.4.1 Interpretation

The interpretation of propositions and contexts from Section 2.4.1 remains the same, except that the interpretation of $\top$ is updated as $[\![\top]\!] = \mathcal{S}$.

The interpretation of the deduction rules is also the same as that from Section 2.4.1, except for the introduction and elimination of $\top$, and the extra rules prod($\mathfrak{s}$), which are the following:

$$\begin{aligned}
[\![\Gamma \vdash \star_{\mathfrak{s}} : \top]\!] &= [\![\Gamma]\!] \xrightarrow{!} \{\star\} \xrightarrow{\mathfrak{s}} \mathcal{S} \\
[\![\Gamma \vdash \delta_{\top}(t,u) : A]\!] &= [\![\Gamma]\!] \xrightarrow{\Delta} [\![\Gamma]\!] \times [\![\Gamma]\!] \xrightarrow{t\times u} \mathcal{S} \times [\![A]\!] \xrightarrow{\bullet_A} [\![A]\!] \\
[\![\Gamma \vdash \mathfrak{s} \bullet t : A]\!] &= [\![\Gamma]\!] \xrightarrow{t} [\![A]\!] \xrightarrow{\hat{\mathfrak{s}}} [\![A]\!]
\end{aligned}$$

3.4.2 Soundness

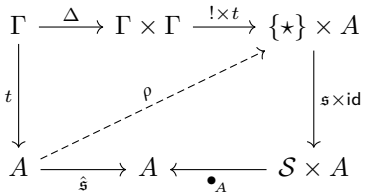
The proof of soundness is updated from the proof of soundness from Section 2.4.2.

As before, we require the substitution lemma (proved in [10, Appendix D.1]).

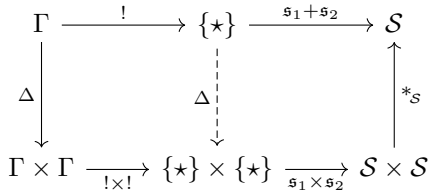
▶ **Lemma 3.8** (Substitution). *If $x : A, \Gamma \vdash t : B$ and $\Gamma \vdash u : A$, then $[\![\Gamma \vdash (u/x)t : B]\!] = [\![\Gamma \vdash t : B]\!] \circ ([\![\Gamma \vdash u : A]\!] \times \mathsf{id}) \circ \Delta$.*

▶ **Theorem 3.9** (Soundness). *If $t \longrightarrow r$ and $\Gamma \vdash t : A$, then $[\![\Gamma \vdash t : A]\!] = [\![\Gamma \vdash r : A]\!]$.*

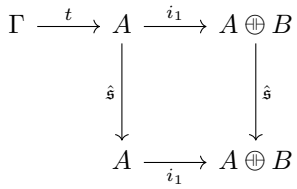
Proof. By induction on the relation $\longrightarrow$. The full proof can be found in [10, Appendix D.2]. We only give a few illustrative examples.
Rule $\delta_{\top}(\star_{\mathfrak{s}}, t) \longrightarrow \mathfrak{s} \bullet t$. The commuting diagram is the following:



The commutation of the left diagram is trivial. The right diagram is the definition of $\hat{\mathfrak{s}}$. Rule $\star_{\mathfrak{s}_1} \parallel \star_{\mathfrak{s}_2} \longrightarrow \star_{\mathfrak{s}_1+\mathfrak{s}_2}$. The commuting diagram is the following:



The left diagram commutes by naturality of Δ. The right diagram commutes since the magma operation of $\mathcal{S}$ is defined with respect to the bi-magma operation: $*_{\mathcal{S}} = +$. Rule $\mathfrak{s} \bullet \mathsf{inl}(t) \longrightarrow \mathsf{inl}(\mathfrak{s} \bullet t)$. The commuting diagram is



The diagram commutes since $\mathfrak{s} \bullet_{\oplus} (a, 0) = (\mathfrak{s} \bullet_A a, 0)$ by definition. ◀

3.4.3 Adequacy

In this setting, the concept of computational equivalence is simpler than in the calculus $\lambda_{\|}$, since the proposition $\top \vee \top$ no longer represents the booleans but rather the action magma $\mathcal{S}^2$. Thus, to distinguish two programs, it is sufficient to use the proposition $\top$, as it contains as many scalars as $\mathcal{S}$ (unless $\mathcal{S} = \emptyset$, in which case the language becomes trivial, or $\mathcal{S} = \{\star\}$, in which case the language is analogous to $\lambda_{\|}$).

▶ **Theorem 3.10** (Adequacy). *If* $[\![\vdash t : A]\!] = [\![\vdash u : A]\!]$, *then* $t \sim u$.

Proof. By induction on the structure of A. See [10, Appendix D.3]. ◀

4 Conclusion

Summary. In this paper, we have proposed a categorical framework for modelling parallelism and algebraic combinations within intuitionistic propositional logic, without relying on monads or biproducts. Our approach is based on the minimal structure of magmas and action magmas, with arrows taken from the category **Set**, which allows us to interpret two lambda calculi: a parallel lambda calculus $\lambda_{\|}$ and its algebraic extension $\lambda^{\mathcal{S}}_{\mathsf{alg}}$.

A central challenge addressed in this work is the interpretation of disjunction in the presence of parallel and scalar operators. Unlike in certain models of linear logic – those equipped with biproducts – where it is possible to identify disjunction and conjunction via a common categorical structure, intuitionistic logic does not provide this feature. Our solution, based on a set-theoretic construction combining the disjoint union and the Cartesian product, allows us to model disjunction while preserving the intended behaviour of parallel composition. This highlights the disjunction as the most delicate connective to handle in this context, and its resolution was crucial to enabling a uniform interpretation of both calculi.

By showing that these effects can be captured without requiring monads or biproducts, we provide an alternative and structurally simpler foundation for reasoning about parallelism in logic. In particular, our work suggests that many of the insights from linear logic – including the idea of interpreting computational effects through categorical structure – can be translated to a broader logical setting, provided that we carefully reconsider the structural assumptions.

Homomorphisms. Neither $\mathbf{Mag}_{\mathbf{Set}}$ nor $\mathbf{AMag}^{\mathcal{S}}_{\mathbf{Set}}$ are algebraic categories, in the sense that the natural arrows for the given objects would be homomorphisms preserving the operations. The reason for this differs for each calculus and deserves some discussion.

In the case of the $\lambda_{\|}$ calculus, we could have opted for a category whose arrows preserve the operations of the objects. However, in such a case, it would not have been sufficient to work with magmas, since the arrow $*_{\oplus} : (A \oplus B) \times (A \oplus B) \longrightarrow A \oplus B$ is not a homomorphism unless it is associative and commutative, which is not the case unless $*_A$ and $*_B$ are. Indeed, if $f = *_{\oplus}$, then $f((c_1, c_2) *_{\times} (c_3, c_4)) = f(c_1 *_{\oplus} c_3, c_2 *_{\oplus} c_4) = (c_1 *_{\oplus} c_3) *_{\oplus} (c_2 *_{\oplus} c_4) \neq (c_1 *_{\oplus} c_2) *_{\oplus} (c_3 *_{\oplus} c_4) = f(c_1, c_2) *_{\oplus} f(c_3, c_4)$. Moreover, the category of commutative magmas is not Cartesian closed, and thus, we would not be able to interpret the $\lambda_{\|}$ calculus in it.

The advantage of using the category $\mathbf{Mag}_{\mathbf{Set}}$ is that it allows us to interpret the parallel operator in a more general setting, where the operator does not need to be associative or commutative. It also gives us a closer connection to the algebraic lambda calculus, where the interpretation cannot be done in a category with homomorphisms. Indeed, it is well

known [2] that $(\lambda x.t)(\mathfrak{a} \bullet u)$ and $\mathfrak{a} \bullet (\lambda x.t)u$ do not yield the same result, unless the linearity is enforced by the reduction strategy as is the case of Lineal [1][1]. An easy counterexample, taking $\mathcal{S} = \mathbb{N}$, is the following: $(\lambda x.\star_1)(2 \bullet t) \longrightarrow \star_1 \neq \star_2 \longleftarrow 2 \bullet \star_1 \longleftarrow 2 \bullet (\lambda x.\star_1)t$.

The interpretation of disjunctions. As mentioned in the introduction, the fact that the parallel of disjunctions can be seen as a pair is evident. For example, compare $\mathsf{inl}(t) \parallel \mathsf{inr}(u)$ with $\langle t, u \rangle$, where t and u are terms of type A and B, respectively. This resemblance becomes even more apparent when considering the Linear Logic additive disjunction and conjunction, which are both interpreted as a biproduct.

The interpretation of disjunction that we gave for the non-linear case, namely $A \oplus B = (A \cup B) \cup (A \times B)$, attempts to mimic this biproduct-like behaviour, but without making disjunction and conjunction coincide. Instead, it establishes an inclusion. Indeed, when linearity is not considered, we cannot simply add a zero to a conjunction to obtain a disjunction. In a conjunction, both t and u are present, whereas in a disjunction with parallel, both t and u can be present – but it is still possible to have only one of them. This provides a more refined interpretation of disjunction.

We stress that our system does feature a proper disjunction: the only novelty is that, due to the presence of the parallel operator, terms such as $\mathsf{inl}(v) \parallel \mathsf{inr}(w)$ may also be viewed as conjunctive. This viewpoint is consistent with the recent preprint [12], where instead of a parallel operator an additional introduction rule $\mathsf{inlr}(v, w)$ is considered, which is equivalent to $\mathsf{inl}(v) \parallel \mathsf{inr}(w)$.

This perspective also highlights the added value of our model: the introduction of the pokeball product provides a semantic account of this phenomenon, which had previously been treated only at the syntactic level. In particular, it captures the situation where both sides of a disjunction are present simultaneously. The pokeball product shows that this behaviour can be modelled categorically in intuitionistic logic, without requiring biproducts, thereby revealing a subtle logical feature of disjunction in the presence of parallelism.

Future directions. The calculi we presented are strongly normalising. As future work, we plan to investigate the addition of fixpoint operators to extend their expressive power while ensuring that desirable semantic properties are preserved.

More broadly, we believe that the approach developed here could be extended to capture other computational effects beyond parallelism and algebraic combinations, in a spirit similar to Moggi's monadic framework. Identifying the appropriate categorical structures for such effects remains an open and promising direction for future work.

References

1 Pablo Arrighi and Gilles Dowek. Lineal: a linear algebraic λ-calculus. *Logical Methods in Computer Science*, 13(1:8):1–33, 2017. doi:10.23638/LMCS-13(1:8)2017.

2 Ali Assaf, Alejandro Díaz-Caro, Simon Perdrix, Christine Tasson, and Benoît Valiron. Call-by-value, call-by-name and the vectorial behaviour of the algebraic λ-calculus. *Logical Methods in Computer Science*, 10(4:8), 2014. doi:10.2168/LMCS-10(4:8)2014.

3 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2007. doi:10.1007/978-3-540-74915-8_24.

[1] In Lineal, $(\lambda x.t)(\mathfrak{a} \bullet u)$ does not beta-reduce, but first reduces to $\mathfrak{a} \bullet (\lambda x.t)u$ until u does not include any more scalars or sums, so, forcing the linearity by evaluation.

4 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. A relational model of a parallel and non-deterministic lambda-calculus. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings*, volume 5407 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-540-92687-0_8.

5 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. A relational semantics for parallelism and non-determinism in a functional setting. *Annals of Pure and Applied Logic*, 163(7):918–934, 2012. doi:10.1016/j.apal.2011.09.008.

6 Horatiu Cirstea, Germain Faure, and Claude Kirchner. A ρ-calculus of explicit constraint application. *Higher-Order and Symbolic Computation*, 20(1-2):37–72, 2007. doi:10.1007/s10990-007-9004-2.

7 Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. A filter model for concurrent λ-calculus. *SIAM Journal on Computing*, 27(5):1376–1419, 1998. doi:10.1137/S0097539794275860.

8 Alejandro Díaz-Caro and Gilles Dowek. A new connective in natural deduction, and its application to quantum computing. *Theoretical Computer Science*, 957:113840, 2023. doi:10.1016/j.tcs.2023.113840.

9 Alejandro Díaz-Caro and Gilles Dowek. A linear linear lambda-calculus. *Mathematical Structures in Computer Science*, 34(10):1103–1137, 2024. doi:10.1017/S0960129524000197.

10 Alejandro Díaz-Caro and Octavio Malherbe. Beyond monads and biproducts: A uniform interpretation of parallelism in intuitionistic logic. Extended version with appendix at arXiv:2408.16102, 2025. doi:10.48550/arXiv.2408.16102.

11 Alejandro Díaz-Caro and Octavio Malherbe. The sup connective in IMALL: A categorical semantics. Draft at arXiv:2205.02142, 2025. doi:10.48550/arXiv.2205.02142.

12 Alejandro Díaz-Caro and Gilles Dowek. A new introduction rule for disjunction. Draft at arXiv:2502.19172, 2025. doi:10.48550/arXiv.2502.19172.

13 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1–3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.

14 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 301–310. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.36.

15 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 2 edition, 1991. doi:10.1007/978-1-4757-4721-8.

16 Giulio Manzonetto. *Models and theories of lambda calculus*. PhD thesis, Univsità Ca'Foscari (Venice) and Université Paris Diderot (Paris 7), 2008. URL: https://tel.archives-ouvertes.fr/tel-00715207.

17 Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. doi:10.1016/S0304-3975(97)00143-6.

18 Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

19 Lionel Vaux. The algebraic lambda calculus. *Mathematical Structures in Computer Science*, 19(5):1029–1059, 2009. doi:10.1017/S0960129509990089.

Beyond Exact Fairness: Envy-Free Incomplete Connected Fair Division

Ajaykrishnan E S
University of California, Santa Barbara, CA, USA

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA

Abstract

We study the problem of Envy-Free Incomplete Connected Fair Division, where exactly p vertices of an undirected graph must be allocated to agents such that each agent receives a connected share and does not envy another agent's share. Focusing on agents with additive valuations, we show that the problem remains computationally hard when parameterized by p and the number of agents. This result holds even for star graphs and with the input numbers given in unary representation, thereby resolving an open problem posed by Gahlawat and Zehavi (FSTTCS 2023).

In stark contrast, we show that if one is willing to tolerate even the slightest amount of envy, then the problem becomes efficient with respect to the natural parameters. Specifically, we design an Efficient Parameterized Approximation Scheme parameterized by p and the number of agent types. Our algorithm works on general graphs and remains efficient even when the input numbers are provided in binary representation.

2012 ACM Subject Classification Theory of computation → W hierarchy; Theory of computation → Fixed parameter tractability

Keywords and phrases Envy-Free Incomplete Connected Fair Division, Efficient Parameterized Approximation Scheme, W[1]-hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.29

Funding NSF Grant No. 2505099, Collaborative Research: AF Medium: Structure and Quasi-Polynomial Time Algorithms.

1 Introduction

Fair division is a critical problem in resource allocation, focused on distributing resources among multiple agents in a way that meets certain fairness criteria. The theoretical study of this problem has been a core area of research, with work dating as far back back as the 1940s [33]. Fair division, beyond its theoretical appeal, has numerous practical applications, as demonstrated by platforms like Spliddit [24], which use fair allocation algorithms to solve real-world problems such as sharing rent, splitting taxi fares, and assigning goods. However, for many of these practical scenarios, one ends up having to address more complex constraints. An example of this is the case where the goods are *indivisible* and the bundle of items corresponding to each agent must satisfy some *connectivity* requirements. The φ-Connected Fair Division (φ-CFD) problem, introduced by Bouveret et al. [11] models this using a graph whose vertices represent items, where the bundles allocated to agents are required to form connected subsets of the vertices, where φ specifies some desired notion of fairness.

However, this model requires that every vertex of the graph be allocated to some agent, which fails to cover some natural scenarios where there is a need to be *economic* in the allocation. As an example consider a museum that plans to allocate sections of a large exhibition hall for various temporary art exhibits. Each exhibit needs a connected space to display its artwork effectively, but the museum must also reserve areas for future events or special exhibits. The challenge then becomes to assign some part of the available space fairly among the groups, ensuring each receives a connected section, while leaving enough space


© Ajaykrishnan E S and Daniel Lokshtanov;
licensed under Creative Commons License CC-BY 4.0

unallocated for other purposes. Another scenario where being economic is crucial involves allocating resource-intensive goods, such as when a company rents office spaces or a startup purchases equipment for its employee teams.

Motivated by these considerations, Gahlawat and Zehavi [21] introduced the φ-Incomplete Connected Fair Division problem (φ-ICFD). In this, the goal is to find a *valid* and φ-allocation, that is, an allocation of exactly p vertices of the graph where each agent receives a distinct connected bundle that satisfies the fairness criterion specified by φ. From the perspective of parameterized complexity, they consider this problem for various well known notions of fairness, one of them being envy-freeness. An allocation of vertices to agents is *envy-free* if, for every agent, the value they assign to their own bundle is at least as large as the value they assign to any other agent's bundle. They show that, if the valuations are provided in binary representation, then the Envy-Free Incomplete Connected Fair Division problem becomes W[1] hard parameterized by p and the number of agents. However, their work leaves behind the following open problem: does Envy-Free Incomplete Connected Fair Division admit a Fixed-Parameter Tractable (FPT) algorithm parameterized by number of agents and p, if the valuations are given in unary representation?

1.1 Our Contribution

We resolve the open problem posed by Gahlawat and Zehavi [21] by proving that Envy-Free Incomplete Connected Fair Division is W[1]-hard when parameterized by $p + |A|$, where A is the set of agents, when the input numbers are in unary representation, even for highly restricted input instances such as star graphs. Our result follows via a reduction from (k, M)-Vector-Sum, a multidimensional variant of the classic NP-hard problem Subset Sum, shown to be W[1]-hard by Abboud et al. [1], parameterized by $k + d$, where d is the dimension of the input vectors, even when the input numbers are in unary representation.

▶ **Theorem 1.1.** Envy-Free Incomplete Connected Fair Division *is W[1]-hard parameterized by* $p + |A|$*, with the input numbers provided in unary representation, even on star graphs.*

To address the intractability implied by Theorem 1.1, we approach the problem from the perspective of approximate fairness. The study of finding *approximately fair* allocations, meaning allocations that are at most a small multiplicative (or additive factor) unfair to the agents, was initiated by Lipton et al. [29] in the context of cake-cutting and has since been investigated in other settings. For a comprehensive overview of this topic, we refer the reader to the survey by Amanatidis et al. [4], which discusses approximate fairness in various settings. In our case, an allocation is *ε-envy-free* if, for every agent, the value they assign to their own bundle scaled up by a multiplicative factor of $(1 + \varepsilon)$, is at least as large as the value they assign to any other agent's bundle.

In this setting, it is very natural to apply a geometric rounding scheme to round the involved numbers to the nearest power of $(1 + \varepsilon)$ and make some guesses regarding the agents valuation of allocated items. This is especially tempting given that the hardness of the instance in the proof of Theorem 1.1 arises precisely from the inability to iterate over all possible valuations that agents may assign to the allocated items. This does in fact lead to a straightforward FPT algorithm, parameterized by $p + |A|$, that finds a valid and ε-envy-free allocation or correctly concludes that the input instance has no valid and envy-free allocation, if the input numbers are provided in unary representation. But, in the case where the input numbers are provided in binary representation, the agents could have wildly different valuations for the same items and consequently the number of guesses

will no longer be FPT in terms of the relevant parameters. In fact, due to this reason, for some time we believed that there exists a positive real number c such that, assuming the Exponential Time Hypothesis [27], no FPT algorithm can either find a valid and c-envy-free allocation or correctly conclude that the input instance admits no valid and envy-free allocation. Surprisingly, this is not the case – the problem admits an algorithm with the aforementioned guarantees that runs in FPT time parameterized by p and $|\mathcal{A}|$, the number of *agent types*, which is a smaller parameter than the number of agents.

▶ **Theorem 1.2.** *There exists an algorithm that, given an instance* $(G, A, \mathcal{U}, p)$ *of* Envy-Free Incomplete Connected Fair Division *with input numbers in binary representation, and a positive real number* ε*, runs in time* $(\frac{1}{\varepsilon}\log\frac{p}{\epsilon})^{\mathcal{O}(|\mathcal{A}|p^2)}(mn)^{\mathcal{O}(1)}$ *and outputs either a valid and* ε*-envy-free allocation* Π*, or correctly states that* $(G, A, \mathcal{U}, p)$ *does not admit a valid and envy-free allocation.*

We now give a sketch of the algorithm asserted in Theorem 1.2. The algorithm begins by applying a simple reduction rule that allows us to upper bound the number of agents by a function of p and $|\mathcal{A}|$. After this, we proceed to apply the standard technique of color coding [3] (see also [17]) to color the vertices of the graph with p colors, and make some guesses regarding the color of the vertices that end up being assigned to each agent.

At this point, we draw inspiration from the polynomial-time algorithm of Gan et al. [22], which addresses the special case where each agent is allocated exactly one item, sometimes referred to as the *house allocation* problem. To illustrate, consider a more restricted setting in which every agent is assigned a unique color and must be allocated exactly one vertex which has the same color. In this setting, for each agent, we consider the set of items they value the most. If there exists an agent such that none of their most-valued items have the same color as them, then these items can be safely discarded, as allocating them to any agent would make this agent envious. On the other hand, if for every agent there exists at least one most-valued item that has the same color as them, then assigning each agent such an item yields an envy-free allocation. The key idea here is that we can either find an envy-free allocation, or restrict the search-space by a significant amount.

Our setting is more technically challenging since we need to work with approximate envy-freeness, enforce connectivity constraints, and accommodate agents that receive multiple items. For this reason, we encode the restriction of the search space using a *target* vector $\bar{\mu} = (\mu_1, \ldots, \mu_{|A|})$, and an allocation *attains* the target for agent i, if the valuation that agent i has for their own bundle is between $(1+\varepsilon)^{\mu_i - 1}$ and $(1+\varepsilon)^{\mu_i}$. The algorithm guesses a set of properties (of which there are only FPT many) of the valid and envy-free allocation we are searching for, sets the target vector so that $(1+\varepsilon)^{\mu_i} \geq pM_i$, where M_i denotes the valuation of agent i's favorite item, and runs a subroutine that searches for a valid and envy-free allocation that respects these guesses.

The subroutine has an outer loop which, following the key idea from the simpler setting, either finds a solution or decreases the target in each iteration. To prove correctness, we must show that if there exists a valid and envy-free allocation that respects the guesses and attains a target that is at most the current one, then the algorithm does one of the following: it either returns a valid and ε-envy-free allocation, or identifies an agent i such that no valid and envy-free allocation that respects the guesses and attains a target that is at most the current one attains the current target for agent i. We design a procedure that achieves this by observing that the guesses constrain the instances to the point that the allocations corresponding to each agent can be searched for independently of each other. Finding such a feasible assignment for each individual agent is done by reducing to a weighted version of the well studied Graph Motif problem, first introduced by Lacroix et al. [28]. This concludes our sketch.

We remark that a subtle but crucial aspect for the correctness of the algorithm is that the feature guessing must be performed outside the subroutine. Additionally, we draw the reader's attention to the fact that we consider two subtly different versions of the problem. In the optional setting, agents may be assigned an empty bundle, whereas in the mandatory setting, each agent must receive a non-empty set of items. Gahlawat and Zehavi [21] study only the mandatory setting. Our results apply to both the optional and mandatory settings. We first establish our results for the optional setting and then show how they extend to the mandatory case.

1.2 Related Work

Fair division of indivisible goods have recieved a considerable amount of attention in the computational social choice literature, an interested reader can refer some of the recent surveys [4, 34]. The work introducing the fair division of *indivisible* items with *connectivity* requirements on the players' bundles was carried out by Bouveret et al. [11]. Their initial results inspired a series of follow-up studies on the model. This body of work includes studies on conditions guaranteeing the existence of fair allocations [8], achieving various fairness notions such as Pareto-optimal allocations [26] and maximin share allocations [25]. Additionally, Aziz et al. [5] examined scenarios where the items to be allocated could be either goods or bads. There has also been work, examining the *price of connectivity* [6], which evaluates the loss in fairness when allocation are required to be connected, when compared to an optimal disconnected allocation. The study of connected fair division where only a subset of the items are to be allocated, was initiated by Gahlawat and Zehavi [21]. Related notions that have been explored in literature include *donating* a few items to achieve fairness [16] and imposing *cardinality constraints* on the number of items of different types that an agent can receive [9].

The concept of finding *approximately fair* allocations that are at most a small multiplicative (or additive factor) unfair to the agents was introduced by Lipton et al. [29] in the concept of cake-cutting and later on by Budish [15] in the context of indivisible goods. Since then, it has been studied across a variety of settings and fairness notions, including envy-freeness [32], Nash social welfare [2], and max–min fairness [23, 20], among others.

The area of parameterized complexity has found notable use in fair division literature, as many fundamental problems in this domain are (unsurprisingly) NP-hard. The works most relevant to ours include that of Deligkas et al. [18], which provides a comprehensive overview of the parameterized complexity of φ-CFD with respect to the natural parameters as well as several structural parameters of the graph for various fairness notions, and that of Gahlawat and Zehavi [21] which does the same for for the φ-ICFD problem. There is a large body of work separate from this [31, 10], which consider fair division problems from a parameterized perspective with the added requirement of Pareto-optimality. The works of Bredereck et al. [13, 12] apply N-fold Integer Programming to solve problems concerning envy-free and Pareto-efficient allocations of items, with the latter even handling instances where input numbers are given in binary representation. Furthermore, the problem of finding *locally* envy-free allocations, where the input includes a graph in which vertices represent agents, and an agent can only envy their neighbors in the graph have also been explored [7, 14].

2 Preliminaries

For a positive integer n, let $[n]$ be the set $\{1, 2, \ldots, n\}$, and $[a, b]$ denote the set of integers $i \in \mathbb{Z}$ that satisfies $i \leq b$ and $i \geq a$. Let $\mathbb{N}_{\geq 0} = \mathbb{N} \cup \{0\}$. For vectors $\bar{v}^{(1)}, \bar{v}^{(2)} \in \mathbb{Z}^n$, we write $\bar{v}^{(1)} \leq \bar{v}^{(2)}$ (respectively, $\bar{v}^{(1)} = \bar{v}^{(2)}$) to mean that $v_i^{(1)} \leq v_i^{(2)}$ (respectively, $v_i^{(1)} = v_i^{(2)}$),

where $v_i^{(1)}$ and $v_i^{(2)}$ denote the i-th coordinates of $\bar{v}^{(1)}$ and $\bar{v}^{(2)}$, for every $i \in [n]$. Furthermore, we use $\bar{v}^{(1)} + \bar{v}^{(2)}$ to denote the vector $\bar{v} \in \mathbb{Z}^n$ that satisfies $v_i = v_i^{(1)} + v_i^{(2)}$ for every $i \in [n]$. We consider finite, simple, undirected graphs. For standard graph terminology we refer to the book by Diestel [19]. We make a slight departure from standard notations by denoting the number of vertices in a graph by m.

An input instance of the Envy-Free Incomplete Connected Fair Division problem consists of a tuple $(G, A, \mathcal{U}, p)$, where,

- G, the *utility graph*, is a graph on m vertices
- $A = [n]$ is the set of *agents*, where $n \in \mathbb{N}$
- $\mathcal{U}$ is an n-tuple of *valuation functions* $\{u_i : V(G) \to \mathbb{N}_{\geq 0}\}_{i \in [n]}$
- $p \in [0, m]$ is a non-negative integer

The vertices of G are referred to as *items*. As is standard in literature, we assume that agents have *additive valuations* and use $u_i(S)$ to denote $\sum_{v \in S} u_i(v)$ for any $S \subseteq V(G)$. An *allocation* of items is a tuple $\Pi = (\pi_1, \ldots, \pi_n)$, where $\pi_i \subseteq V(G)$ is referred to as the *bundle* of agent i. When clear from the context, we sometimes abuse notations and use π to denote $\bigcup_{i \in A} \pi_i$, the set of allocated vertices. An allocation is *valid* if it allocates exactly p vertices in total, with each bundle being connected and no vertex allocated to multiple bundles. An allocation is *envy-free* if no agent desires another agent's bundle more than their own. Formally, an allocation that is both valid and envy-free must satisfy,

- $\sum_{i=1}^{n} |\pi_i| = p$
- $G[\pi_i]$ is connected
- $\pi_i \cap \pi_j = \emptyset$ for every pair of distinct agents i, j
- $u_i(\pi_i) \geq u_i(\pi_j)$ for every pair of agents i, j

Similarly, an allocation is *ε-envy-free*, for some positive real number ε, if it satisfies $(1 + \varepsilon)u_i(\pi_i) \geq u_i(\pi_j)$ for every agent i and j. Given an instance $\mathcal{I} = (G, A, \mathcal{U}, p)$, the Envy-Free Incomplete Connected Fair Division problem asks whether there exists an allocation in $\mathcal{I}$ that is both valid and envy-free. Here we remark that there are two subtly different versions of the problem. In the *mandatory* setting, each agent must be assigned a non-empty bundle; that is, $|\pi_i| > 0$ for every agent $i \in A$. In contrast, the *optional* setting allows agents to receive empty bundles, meaning $|\pi_i| \geq 0$ for all $i \in A$. Throughout this paper, we use the shorthand EF-ICFD to refer to the envy-free incomplete connected fair division problem in the optional setting.

Two agents i, j have the same *type* if $u_i(v) = u_j(v)$ for every $v \in V(G)$. We use $\mathcal{A}_{\mathcal{I}}$ to denote the set of all agent types in an instance $\mathcal{I}$ of EF-ICFD. Thus $\mathcal{A}_{\mathcal{I}}$ is a partition of A such that two agents i, j belong to the same part $\boldsymbol{a} \in \mathcal{A}_{\mathcal{I}}$ if and only if they have the same type. For ease of notation, we shall drop the subscript $\mathcal{I}$ whenever it is clear from the context. We say that an instance $(G, A, \mathcal{U}, p)$ of EF-ICFD is *in unary* (respectively, *in binary*) if the valuation functions in $\mathcal{U}$ are encoded in unary (respectively, binary). This means there exists a polynomial (respectively, exponential) function f such that $\max\{u_i(v) \mid i \in A, v \in V(G)\} \leq f(|V(G)|)$.

For standard terminology in parameterized algorithms, we refer to the book Parameterized Algorithms [17] by Cygan et al. An *(n, k)-perfect hash family* [30] is a family of functions from $[n]$ to $[k]$ such that for every $S \subseteq [n]$ with $|S| = k$, there exists a function in the family that is injective on S. The following is known regarding (n, k)-perfect hash families.

▶ **Proposition 2.1** ([30], [17]). *For any $n, k \in \mathbb{N}$, one can construct an (n, k)-perfect hash family of size $e^k k^{\mathcal{O}(\log k)} \log n$ in time $e^k k^{\mathcal{O}(\log k)} n \log n$.*

An *Efficient Parameterized Approximation Scheme* (EPAS) for an optimization problem Π with input size n and parameter k is an algorithm that, for every $\varepsilon > 0$, runs in time $f(k, 1/\varepsilon) \cdot n^{\mathcal{O}(1)}$ for some computable function f, and returns a $(1+\varepsilon)$-approximate solution (or $(1-\varepsilon)$-approximate solution for maximization problems) to Π.

3 W[1] Hardness of EF-ICFD Parameterized by p and |A|

To show W[1]-hardness of EF-ICFD parameterized by p and $|A|$, we reduce from (k, M)-Vector-Sum, a multidimensional version of the classic NP-hard problem Subset Sum. We begin by formally defining the problem and stating a known result regarding its complexity.

▶ **Definition 3.1.** *For positive integers k, n, M, and d, the (k, M)-Vector-Sum problem is defined as follows: given a tuple $(\mathcal{W}, \bar{t}, k)$, where $\mathcal{W} = \{w^{(i)}\}_{i\in[n]} \subseteq [0, kM]^d$, and $\bar{t} \in [0, kM]^d$, determine whether there exists a subset $S \subseteq [n]$ of cardinality k, such that $\sum_{i\in S} w^{(i)} = \bar{t}$.*

▶ **Proposition 3.2** (Lemma 4.1 - [1])**.** *(k, M)-Vector-Sum is W[1]-hard parameterized by $k + d$ even when $M = k\, n^{1+o(1)}$.*

With the definition of (k, M)-Vector-Sum in place, we are now prepared to describe the reduction.

3.1 The Reduction

Let $\mathcal{I}_{\text{vs}} = (\mathcal{W}, \bar{t}, k)$ be an instance of $(k, n^{1+o(1)})$-Vector-Sum with d dimensions. We define the following instance $\mathcal{I}_{\text{FD}} = (G, A, \mathcal{U}, p)$ of EF-ICFD:

- Let G be a star with center c, and $2d + n$ leaves. We assign a unique label to each leaf using elements of the set $V_\alpha \cup V_\beta \cup \{v_1, v_2, \ldots, v_n\}$, where $V_\alpha = \{a_1, a_2, \ldots, a_d\}$ and $V_\beta = \{b_1, b_2, \ldots, b_d\}$. With a slight abuse of notation, we use the label to refer to the corresponding vertex.
- Let $A = [2d + 1]$. We assign a unique label to each agent using elements of the set $\{A_1, A_2, \ldots, A_d\} \cup \{B_1, B_2, \ldots, B_d\} \cup \{C\}$. Here as well, we slightly abuse notation, and use the label to refer to the corresponding agent.
- $p = 2d + k + 1$.
- For every $l \in [d]$, we define $u_{A_l}(x) = \begin{cases} n^3 + w_i^{(l)} & x = v_i \\ n^3 & x = c \\ 0 & x \in V_\beta \\ 0 & x \in V_\alpha \setminus \{a_l\} \\ (k+1)n^3 + t_l & x = a_l \end{cases}$
- For every $l \in [d]$, we define $u_{B_l}(x) = \begin{cases} n^3 - w_i^{(l)} & x = v_i \\ n^3 & x = c \\ 0 & x \in V_\alpha \\ 0 & x \in V_\beta \setminus \{b_l\} \\ (k+1)n^3 - t_l & x = b_l \end{cases}$
- $u_C(v) = 1 \quad$ for every $v \in V(G)$

3.2 Proof of Correctness

We shall prove that $(G, p, A, \mathcal{U})$ has a valid and envy-free allocation if and only if $(\mathcal{W}, \bar{t}, k)$ has a subset $S \subseteq [n]$ of cardinality k, such that $\sum_{i \in S} w^{(i)} = \bar{t}$. To this end, we make some observations about the properties that any valid and envy-free allocation in the instance $\mathcal{I}_{\text{FD}}$ must satisfy. Note that since the graph is a star and since the bundles must be connected, only the agent who is allocated the center of the star gets more than one vertex. Furthermore, every agent must receive at least one vertex. Otherwise, as each agent has zero valuation for at most $2d - 1$ vertices, and since a total of $2d + k + 1$ vertices must be allocated, any agent with an empty bundle would be envious of some other agent. Hence, for exactly $2d + k + 1$ vertices to be assigned in total, one agent must receive $k + 1$ and the others must receive one vertex each. Now, since C values every vertex equally, they must receive exactly $k + 1$ vertices; otherwise, they would envy the agent who has more than one vertex. With this, we are ready to prove a key property of this instance.

▶ **Lemma 3.3.** *In any valid and envy-free allocation* Π *in* $(G, p, A, \mathcal{U})$*, every agent except* C *must be allocated the vertex they value the most.*

Proof. Let us consider agent C. Observe that, if π_C contains a vertex a_l for some $l \in [d]$, then it cannot be assigned to A_l, since Π is valid. Therefore, irrespective of which vertex is assigned to A_l, we have $u_{A_l}(\pi_C) \geq (k+1)n^3 > u_{A_l}(\pi_{A_l})$, which contradicts the assumption that Π is envy-free. Hence π_C must not contain any vertex from the set V_α and by a similar argument from V_β. Now observe that, according to any agent X distinct from C, we have $u_X(\pi_C) \geq (k+1)n^3 - k\, n^{1+o(1)} \geq kn^3$. Therefore, for X to not be envious of C, we must have $u_X(\pi_X) \geq kn^3$. But this can only be achieved if X is assigned the vertex they value the most. Hence, in the allocation Π, agent A_l must receive a_l and agent B_l must receive b_l for every $l \in [d]$. This concludes the proof of the lemma. ◀

This gives us all the tools we need to prove the main lemma.

▶ **Lemma 3.4.** *The instance* $(G, p, A, \mathcal{U})$ *admits a valid and envy-free allocation* Π *if and only if there exists a subset* $S \subseteq [n]$ *of cardinality* k *such that* $\sum_{i \in S} w^{(i)} = \bar{t}$ *in the instance* $(\mathcal{W}, \bar{t}, k)$*.*

Proof. Let Π be a valid and envy-free allocation in $(G, p, A, \mathcal{U})$. According to lemma 3.3, in Π, agent A_l and B_l must be allocated a_l and b_l respectively, for every $l \in [d]$. Furthermore, we also have that agent C must be allocated exactly $k + 1$ vertices. Let these vertices be $\{c, v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. We claim that the set $S = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ satisfies $\sum_{i \in S} w^{(i)} = \bar{t}$ in the instance $(\mathcal{W}, \bar{t}, k)$. Observe that, as Π is envy-free, $u_{A_l}(\pi_C)$ is at most $u_{A_l}(\pi_{A_l})$ and $u_{B_l}(\pi_C)$ is at most $u_{B_l}(\pi_{B_l})$. This implies that,

$$n^3 + \sum_{j=1}^{k}(n^3 + w_l^{(i_j)}) \leq (k+1)n^3 + t_l \implies \sum_{j=1}^{k} w_l^{(i_j)} \leq t_l$$

$$n^3 + \sum_{j=1}^{k}(n^3 - w_l^{(i_j)}) \leq (k+1)n^3 - t_l \implies \sum_{j=1}^{k} w_l^{(i_j)} \geq t_l$$

Hence if $(G, p, A, \mathcal{U})$ has a valid and envy-free allocation, then there exists a set $S = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ that satisfies $\sum_{i \in S} w^{(i)} = \bar{t}$ in the instance $(\mathcal{W}, \bar{t}, k)$.

Let $S = \{i_1, i_2, \ldots, i_k\} \subseteq [n]$ be a set of cardinality k, such that, $\sum_{i \in S} w^{(i)} = \bar{t}$ in the instance $(\mathcal{W}, \bar{t}, k)$. Consider an allocation Π which allocates $\{c, v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ to C, a_l to A_l, and b_l to B_l for every $l \in [d]$. We claim that Π is a valid and envy-free

allocation in $(G, p, A, \mathcal{U})$. Observe that, for every agent $X \neq C$, we have, $u_C(\pi_X) = 1 < (k+1) = u_C(\pi_C)$. Furthermore, for every $l \in [d]$ and agent $X \neq A_l$, we have $u_{A_l}(\pi_X) \leq (k+1)n^3 + \sum_{j=1}^{k} w_l^{(i_j)} = (k+1)n^3 + t_l = u_{A_l}(\pi_{A_l})$. Finally, for every $l \in [d]$ and agent $X \neq B_l$, we have $u_{B_l}(\pi_X) \leq (k+1)n^3 - \sum_{j=1}^{k} w_l^{(i_j)} = (k+1)n^3 - t_l = u_{B_l}(\pi_{B_l})$. Consequently, Π is envy-free for the instance $(G, p, A, \mathcal{U})$. Since Π allocates a total of p vertices as disjoint, connected subsets to the agents by definition, its validity follows. This completes the proof of the lemma. ◀

By combining Proposition 3.2 and Lemma 3.4 with the properties of the reduction, we obtain Theorem 1.1

▶ **Theorem 1.1.** Envy-Free Incomplete Connected Fair Division *is W[1]-hard parameterized by* $p + |A|$*, with the input numbers provided in unary representation, even on star graphs.*

Proof. Given an instance $\mathcal{I}_{\text{vs}} = (\mathcal{W}, \bar{t}, k)$ of $(k, n^{1+o(1)})$-Vector-Sum with d dimensions, let $\mathcal{I}_{\text{FD}} = (G, A, \mathcal{U}, p)$ be an instance of EF-ICFD constructed according to Subsection 3.1. Observe that in our construction G is a star graph, $p = 2d + k + 1$ and $|A| = 2d + 1$. Furthermore, the range of all valuation functions are bounded above by $\mathcal{O}(n^3)$ which implies that the instance is in unary and that $|\mathcal{I}_{\text{FD}}| = n^{\mathcal{O}(1)}$. Finally, given $\mathcal{I}_{\text{vs}}$, we can construct $\mathcal{I}_{\text{FD}}$ in $n^{\mathcal{O}(1)}$ time. Therefore, the statement of the theorem follows from Proposition 3.2 and Lemma 3.4. ◀

We conclude by remarking that this hardness proof extends to the mandatory setting without modification, as the properties of the constructed instance ensure that, in any envy-free allocation, every agent is assigned a non-empty set. Consequently, both directions of the proof remain valid even in this context.

4 Approximation Scheme for EF-ICFD Parameterized by p and $|\mathcal{A}|$

We begin this section by stating and proving the reduction rule that helps us upper bound the number of agents in an instance of EF-ICFD by a function of p and $|\mathcal{A}|$.

▶ **Lemma 4.1.** *There exists an algorithm that takes as input an instance* $\mathcal{I} = (G, A, \mathcal{U}, p)$ *of* EF-ICFD*, runs in* $(mn)^{\mathcal{O}(1)}$ *time, and outputs an instance* $\mathcal{I}' = (G', A', \mathcal{U}', p')$ *of* EF-ICFD *such that* $\mathcal{I}'$ *admits a valid and envy-free allocation if and only if* $\mathcal{I}$ *admits one, and furthermore, the number of agents of type* $\boldsymbol{a}$ *in* $\mathcal{I}'$ *is at most* $p' + 1$*, for every* $\boldsymbol{a} \in \mathcal{A}_{\mathcal{I}'}$*.*

Proof. We define $G' := G$, $p' := p$, and $\mathcal{U}' := \mathcal{U}$. To define A', consider each agent type $\boldsymbol{a} \in \mathcal{A}_{\mathcal{I}}$. If there are at most $p + 1$ agents of type $\boldsymbol{a}$ in A, we include all of them in A'. Otherwise, we include an arbitrary selection of $p + 1$ agents of type $\boldsymbol{a}$ in A'. It is clear from definition of $\mathcal{I}'$ that the number of agents of type $\boldsymbol{a}$ in $\mathcal{I}'$ is at most $p' + 1$, for every $\boldsymbol{a} \in \mathcal{A}_{\mathcal{I}'}$. Hence to prove the lemma, it suffice to prove that $\mathcal{I}'$ admits a valid and envy-free allocation if and only if $\mathcal{I}$ admits one.

To this end, let Π' be a valid and envy-free allocation in $\mathcal{I}'$. We define an allocation Π in $\mathcal{I}$, where for agent $i \in A$, we have $\pi_i := \pi'_i$ if $i \in A' \cap A$ and $\pi_i := \emptyset$ otherwise. Since the validity of Π follows directly from that of Π' we are left with proving envy-freeness. Note that no agent is envious of agents in $A \setminus A'$, since they are allocated the empty bundle. Furthermore, agents in $A \cap A'$ are not envious of each other since Π' is an envy-free allocation. Finally, agents of $A \setminus A'$ are not envious of agents in $A \cap A'$, since corresponding to an agent $i \in A \setminus A'$, there exists an agent $j \in A \cap A'$ who has the same type and is not envious of anyone in $A \cap A'$. Therefore, we conclude that Π is a valid and envy-free allocation in $\mathcal{I}$.

Now, let Π be a valid and envy-free allocation in $\mathcal{I}$. Let $\sigma : A' \to A$ be an injective function with the property that $\sigma(i)$ and i are of the same type, and every agent in A who is allocated a non-empty bundle under Π lies in the image of σ. Such a function exists because at most p agents receive non-empty bundles under Π, and A' includes all agents from types with at most $p+1$ agents, as well as at least $p+1$ agents from types that has more than $p+1$ agents in A. We define an allocation Π' in $\mathcal{I}'$ by setting $\pi'_i := \pi_{\sigma(i)}$ for each $i \in A'$. Since Π is a valid and envy-free allocation in $\mathcal{I}$, and Π' is obtained by relabeling agents via a injection that preserves types, it follows that Π' is also valid and envy-free in $\mathcal{I}'$. This concludes the proof of the lemma. ◀

▶ **Observation 4.2.** *If $\mathcal{I} = (G, A, \mathcal{U}, p)$ is an instance of* EF-ICFD *obtained after applying the algorithm of Lemma 4.1, then $|A| \leq (p+1) \cdot |\mathcal{A}|$*

▶ **Observation 4.3.** *If $\mathcal{I}'$ is the instance of* EF-ICFD *obtained after applying the algorithm of Lemma 4.1 to $\mathcal{I}$, then given a valid and envy-free allocation Π' for $\mathcal{I}'$, we can construct one for $\mathcal{I}$ in $(mn)^{\mathcal{O}(1)}$ time.*

Observation 4.2 follows from the fact that each agent is associated with a type in $\mathcal{A}$, and the lemma guarantees that no type in the output instance has more than $p+1$ agents. Observation 4.3 follows from the construction described in the proof of Lemma 4.1, which shows how to obtain a valid and envy-free allocation for the original instance from one for the output instance.

Now, we state a lemma that serves as a subroutine in the algorithm. Since it is not central to the main discussion, we defer its proof to the end of the section.

▶ **Lemma 4.4.** *There exists an algorithm that, given a graph G, a weight function $w : V(G) \to \mathbb{N}_{\geq 0}$, and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of $V(G)$ into k parts, runs in time $3^k \cdot |V(G)|^{\mathcal{O}(1)}$ and outputs a connected subgraph H of G that maximizes $\sum_{v \in V(H)} w(v)$ subject to the constraint that $|V(H) \cap C_i| = 1$ for every $i \in [k]$, if such a subgraph exists, and correctly reports that no such subgraph exists otherwise.*

Now, let $\mathcal{I} = (G, A, \mathcal{U}, p)$ be an instance of EF-ICFD and ε be a positive real number. We make the following definitions corresponding to $\mathcal{I}$ and ϵ. Let $\varepsilon'_\varepsilon > 0$ denote the positive root of the equation $x^2 + 3x = \varepsilon$, and define $q_\varepsilon := 1 + \varepsilon'_\varepsilon$ and $t_\varepsilon := 1 + \left\lceil \log_{q_\varepsilon} \left(\frac{p}{\varepsilon'_\varepsilon} \right) \right\rceil$. Finally, for positive real number ρ and $\alpha \in \mathbb{N}_{\geq 0}$, let $\overline{\log}_\rho \alpha$ denote $\lceil \log \alpha \rceil$, if $\alpha > 0$ and -1 otherwise.

▶ **Definition 4.5.** *We define a* bucket profile *to be a tuple $(\mathcal{C}, \mathcal{S}, \Lambda)_{\varepsilon,\mathcal{I}}$, where $\mathcal{C} = \{C_c\}_{c \in [p]}$ is a partition of $V(G)$, $\mathcal{S} = \{S_i\}_{i \in A}$ is a partition of $[p]$, and $\Lambda : A \times [p] \to [0, t_\varepsilon]$ is a function such that, for every pair of distinct agents $i, j \in A$, we have $\sum_{\substack{c \in S_j \\ \Lambda(i,c) \neq t_\varepsilon}} q_\varepsilon^{-\Lambda(i,c)} \leq q_\varepsilon$.*

Referring back to the high-level sketch of the algorithm in Section 1, we note that the set $\mathcal{C}$ corresponds to the partitioning obtained through color coding, $\mathcal{S}$ captures the guess that maps colors to agents, and Λ encodes the guessed features of the allocation. In particular, $\Lambda(i, c)$ represents the guess on how far apart (in terms of powers of q) the valuation of a vertex in color class c and the valuation of the bundle allocated to agent i is, with respect to u_i, in the valid and envy-free allocation we are searching for. The following definition formalizes what it means for an allocation to be consistent with the guesses.

▶ **Definition 4.6.** *An allocation Π in $\mathcal{I}$* respects *a bucket profile $(\mathcal{C}, \mathcal{S}, \Lambda)_{\varepsilon,\mathcal{I}}$ if it satisfies the following:*

- *For every agent $i \in A$ and $c \in [p]$, we have $|\pi_i \cap C_c| = 1$ if $c \in S_i$, and $|\pi_i \cap C_c| = 0$ otherwise;*

- *For every agent $i \in A$, $c \in [p]$, and $v \in \pi \cap C_c$, we have*
  - $\overline{\log}_{q_\varepsilon}(u_i(\pi_i)) - \overline{\log}_{q_\varepsilon}(u_i(v)) \geq \Lambda(i,c)$ *or* $u_i(v) = 0$ *if* $\Lambda(i,c) = t_\varepsilon$*, and,*
  - $\overline{\log}_{q_\varepsilon}(u_i(\pi_i)) - \overline{\log}_{q_\varepsilon}(u_i(v)) = \Lambda(i,c)$ *and* $u_i(v) > 0$ *otherwise.*

Finally, we formally define the target vector and what it means for an allocation to be consistent with it.

▶ **Definition 4.7.** *Let* $M := \max\{u_i(v) \mid v \in V(G), i \in A\}$*. We define a* target vector $\bar{\mu}_{\varepsilon,\mathcal{I}} \in \left[-1, \lceil \log_{q_\varepsilon}(pM) \rceil\right]^A$ *and use* μ_i *to denote the entry of* $\bar{\mu}_{\varepsilon,\mathcal{I}}$ *associated with agent* $i \in A$*. An allocation* Π attains *the target* $\bar{\mu}_{\varepsilon,\mathcal{I}}$ *for agent* $i \in A$*, if* $\overline{\log}_{q_\varepsilon}(u_i(\pi_i)) = \mu_i$*.*

For the rest of the discussion, for ease of notation, we drop the subscript $\mathcal{I}$ and ε, whenever it is clear from context.

The following lemma is the main subroutine of the algorithm.

▶ **Lemma 4.8.** *There exists an algorithm that, given an instance* $(G, A, \mathcal{U}, p)$ *of* EF-ICFD*, a positive real number* ε*, a bucket profile* $(\mathcal{C}, \mathcal{S}, \Lambda)$*, and a target vector* $\bar{\mu}$*, runs in time* $3^p \cdot m^{\mathcal{O}(1)}$ *and returns one of the following:*

- *A valid and* ε*-envy-free allocation* Π*, or*
- *An agent* $i \in A$ *such that no valid and envy-free allocation that respects the bucket profile* $(\mathcal{C}, \mathcal{S}, \Lambda)$ *attains a target vector* $\bar{\mu}'$*, where* $\bar{\mu}' \leq \bar{\mu}$ *with* $\mu'_i = \mu_i$*.*

Proof. We define vertex $v \in V(G)$ to be *eligible* if, letting $c \in [p]$ be such that $v \in C_c$, for every agent $i \in A$ with $c \notin S_i$, we have $\overline{\log}_q(u_i(v)) \leq \mu_i - \Lambda(i,c)$ or $u_i(v) = 0$. Let G_i be the subgraph of G induced by the eligible vertices in $\bigcup_{c \in S_i} C_c$. We define a subgraph H_i of G_i to be *colorful*, if $|V(H_i) \cap C_c| = 1$ for every $c \in S_i$. Before describing the algorithm, we prove the following claims:

▷ Claim 4.9. If for every agent $i \in A$ there exists a connected, colorful subgraph H_i of G_i satisfying the inequality $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$, then the allocation Π defined by $\pi(i) := V(H_i)$ is a valid and ε-envy-free allocation.

Proof. We begin by observing that $\{\pi_i\}_{i \in A}$ are pairwise disjoint. Otherwise, if there exists a vertex $v \in \pi_i \cap \pi_j$ for some pair of distinct agents $i, j \in A$, then $v \in V(G_i) \cap V(G_j)$ which implies that either v belongs to two distinct sets in $\mathcal{C}$, which contradicts the fact that $\mathcal{C}$ is a partition of $V(G)$, or the set C_c containing v belongs to both S_i and S_j, which contradicts the fact that $\mathcal{S}$ is a partition of $[p]$. Moreover, we have $\sum_{i \in A} |\pi_i| = \sum_{i \in A} |S_i| = p$, since $\mathcal{S}$ is a partition of $[p]$. Finally, as each $G[\pi_i]$ is connected by definition, it follows that Π is a valid allocation.

Now, to prove ε-envy-freeness, consider $c \in [p]$, agent $i \in A$ such that $c \notin S_i$, and vertex $v \in \pi \cap C_c$. If $u_i(v)$ is positive, then $\overline{\log}_q(u_i(v)) = \lceil \log_q(u_i(v)) \rceil \geq 0$. Moreover, since $\Lambda(i,c)$ is non-negative and v is eligible, it follows that $0 \leq \mu_i \leq \overline{\log}_q(u_i(\pi_i)) = \lceil \log_q(u_i(\pi_i)) \rceil$. Therefore, whenever $u_i(v)$ is positive, we have the following inequality, :

$$\begin{aligned} u_i(v) \cdot q^{\Lambda(i,c)} &\leq u_i(v) \cdot q^{\mu_i - \overline{\log}_q(u_i(v))} \\ &\leq u_i(v) \cdot q^{\log_q(u_i(\pi_i)) + 1 - \log_q(u_i(v))} \\ &= u_i(\pi_i) \cdot q \end{aligned} \tag{1}$$

Thus, if $i, j \in A$ are distinct agents, then,

$$\begin{aligned} u_i(\pi_j) \quad = \quad \sum_{\substack{c \in S_j \\ v \in \pi_j \cap C_c}} u_i(v) \quad &\leq \quad \sum_{\substack{c \in S_j \\ \Lambda(i,c)=t}} u_i(\pi_i) \cdot q^{1-\Lambda(i,c)} + \sum_{\substack{c \in S_j \\ \Lambda(i,c) \neq t}} u_i(\pi_i) \cdot q^{1-\Lambda(i,c)} \\ &\leq \quad p \cdot u_i(\pi_i) \cdot q^{1-t} \; + \; q^2 u_i(\pi_i) \\ &\leq \quad \varepsilon' u_i(\pi_i) \; + \; (1+\varepsilon')^2 u_i(\pi_i) \\ &\leq \quad (1+\varepsilon) u_i(\pi_i) \end{aligned}$$

where the first transition follows from the fact that π_j contains exactly one vertex from each C_c for $c \in S_j$; the second follows from inequality (1); the third uses the observation that there are at most p elements $c \in S_j$ with $\Lambda(i,c) = t$ and the assumption that $(\mathcal{C}, \mathcal{S}, \Lambda)$ is a bucket profile, which implies $\sum_{\substack{c \in S_j \\ \Lambda(i,c) \neq t}} q^{-\Lambda(i,c)} \leq q$; the fourth and fifth follow from the definitions of q, t, and ε'. Therefore, Π is ε-envy-free, which completes the proof of the claim. ◁

▷ Claim 4.10. If there exists an agent $i \in A$ such that no connected, colorful subgraph H_i of G_i satisfies $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$, then there does not exist a valid and envy-free allocation that respects $(\mathcal{C}, \mathcal{S}, \Lambda)$ and attains a target vector $\bar{\mu}'$, where $\bar{\mu}'$ is a target vector that satisfies $\bar{\mu}' \leq \bar{\mu}$ and $\mu'_i = \mu_i$.

Proof. Let $i \in A$ be an agent such that no connected, colorful subgraph H_i of G_i satisfies $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$. Let $\bar{\mu}'$ be a target vector that satisfies $\bar{\mu}' \leq \bar{\mu}$ and $\mu'_i = \mu_i$. Suppose for contradiction that there exists a valid and envy-free allocation Π that respects $(\mathcal{C}, \mathcal{S}, \Lambda)$, and attains $\bar{\mu}'$.

Since Π is valid, $G[\pi_i]$ is connected. Moreover, as Π attains $\bar{\mu}'$, we have $\overline{\log}_q(u_i(\pi_i)) = \mu'_i = \mu_i$. Furthermore, observe that since Π respects $(\mathcal{C}, \mathcal{S}, \Lambda)$, each vertex in π_i belongs to a distinct set in $\mathcal{C}$ whose indices belong to S_i, which ensures that $G[\pi_i]$ is colorful. Finally, to show that $G[\pi_i]$ is a subgraph of G_i, note that for every $c \in S_i$, $v \in C_c$, and agent $j \in A$ that is distinct from i, if $u_j(v) > 0$ then we have $\overline{\log}_q(u_j(\pi_j)) - \overline{\log}_q(u_j(v)) \geq \Lambda(j,c)$. As Π attains $\bar{\mu}'$, it follows that $\overline{\log}_q(u_j(\pi_j)) = \mu'_j \leq \mu_j$, which implies $\overline{\log}_q(u_j(v)) \leq \mu_j - \Lambda(j,c)$. Hence, every vertex in π_i is eligible, and consequently, $G[\pi_i]$ is a subgraph of G_i. Hence, $G[\pi_i]$ is a connected, colorful subgraph of G_i satisfying $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$, which contradicts the definition of i. This completes the proof of the claim. ◁

Algorithm. For each agent $i \in A$, the algorithm computes the graph G_i, defined to be the subgraph of G induced by the eligible vertices in $\bigcup_{c \in S_i} C_c$. It calls the algorithm of Lemma 4.4 with input $(G_i, u_i, \{C_c\}_{c \in S_i})$ as a subroutine to compute a maximum-weight connected, colorful subgraph H_i of G_i. If for every agent $i \in A$ a subgraph H_i is found, for which $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$, then the algorithm returns the allocation Π defined by $\pi(i) := V(H_i)$. Otherwise, it returns an agent $i \in A$ for which no connected, colorful subgraph satisfying $\overline{\log}_q(u_i(V(H_i))) \geq \mu_i$ could be found.

The correctness of the algorithm follows from Claims 4.9 and 4.10, along with the correctness of the subroutine. The running time bound follows from the fact that the subroutine takes $3^p m^{\mathcal{O}(1)}$ time, along with the observation that all other steps of the algorithm run in $m^{\mathcal{O}(1)}$ time. This completes the proof of Lemma 4.8. ◀

With this we are ready to prove Theorem 1.2.

▶ **Theorem 1.2.** *There exists an algorithm that, given an instance* $(G, A, \mathcal{U}, p)$ *of* Envy-Free Incomplete Connected Fair Division *with input numbers in binary representation, and a positive real number* ε*, runs in time* $(\frac{1}{\varepsilon}\log\frac{p}{\epsilon})^{\mathcal{O}(|\mathcal{A}|p^2)}(mn)^{\mathcal{O}(1)}$ *and outputs either a valid and* ε*-envy-free allocation* Π*, or correctly states that* $(G, A, \mathcal{U}, p)$ *does not admit a valid and envy-free allocation.*

Proof. If the instance $(G, A, \mathcal{U}, p)$ satisfies $|A| > (p+1)|\mathcal{A}|$, then we apply the algorithm from Lemma 4.1 to obtain a new instance $\mathcal{I}'$, for which we have $|A_{\mathcal{I}'}| \leq (p+1)|\mathcal{A}_{\mathcal{I}'}|$, by Observation 4.2. We then solve $\mathcal{I}'$ and, using Observation 4.3 and Lemma 4.1, reconstruct a solution for the original instance $(G, A, \mathcal{U}, p)$, incurring only a negligible overhead of $m^{\mathcal{O}(1)}$ time. Therefore, for the remainder of the proof, we may assume that we are working with an instance that satisfies $|A| \leq (p+1)|\mathcal{A}|$.

Now, let us fix an arbitrary labeling l on $V(G)$ using the set $[m]$. We say that a partition $\mathcal{C}$ of $V(G)$ is induced by a function $f : [m] \to [p]$, if for every $v \in V(G)$, we have that $v \in C_c$ if and only if $f(l(v)) = c$, where $l(v)$ denotes the label of v. Let $\mathcal{B}$ denote the set of all bucket profiles that belong to the set $\mathcal{F}_{\mathcal{C}} \times \mathcal{F}_{\mathcal{S}} \times \mathcal{F}_{\Lambda}$, where $\mathcal{F}_{\mathcal{C}}$ is the set of partitions of $V(G)$ induced by elements of an (m, p)-perfect hash family of size $e^p p^{\mathcal{O}(\log p)} \log m$ whose existence is guaranteed by Proposition 2.1, $\mathcal{F}_{\mathcal{S}}$ is the set of all partitions of the set $[p]$ into n parts, where some parts may be empty, and $\mathcal{F}_{\Lambda}$ is the set of all functions from $A \times [p]$ to $[0, t]$. Let $M := \max\{u_i(v) \mid v \in V(G),\, i \in A\}$, and define a target vector $\bar{\mu}' = (\mu'_1, \ldots, \mu'_n)$, where $\mu'_i := \lceil \log_q(p \cdot M) \rceil$ for each $i \in A$.

Algorithm. For each bucket profile in $\mathcal{B}$, the algorithm initializes the target vector $\bar{\mu}$ to $\bar{\mu}'$. It then invokes the algorithm from Lemma 4.8 as a subroutine. If it returns a valid and ε-envy-free allocation Π, the algorithm outputs Π and terminates. Otherwise, upon receiving an agent $i \in A$, it updates $\bar{\mu}$ by reducing μ_i by one and iterates by invoking the algorithm from Lemma 4.8 again. This loop continues until some entry of $\bar{\mu}$ becomes equal to negative two, at which point the algorithm proceeds to the next bucket profile in $\mathcal{B}$. If no allocation is found after all elements of $\mathcal{B}$ have been considered, the algorithm concludes that the instance $(G, A, \mathcal{U}, p)$ admits no valid and envy-free allocation.

To prove correctness of the algorithm, assume that $(G, A, \mathcal{U}, p)$ is an instance of EF-ICFD that has a valid and envy-free allocation Π. We only need to show that the algorithm returns an allocation, since by Lemma 4.8, any allocation that is returned by the subroutine and consequently the algorithm will be valid and ε-envy-free. Toward this end, we make the following claim.

▷ Claim 4.11. If $(G, A, \mathcal{U}, p)$ is an instance of EF-ICFD that has a valid and envy-free allocation Π, then there exists a bucket profile $(\mathcal{C}, \mathcal{S}, \Lambda) \in \mathcal{B}$ that is respected by Π.

Proof. Let $\mathcal{C}$ be an element of $\mathcal{F}_{\mathcal{C}}$, such that for every $c \in [p]$, we have that $|C_c \cap \bigcup_{i \in A} \pi_i| = 1$. To see that such an element exists, observe that since Π is a valid allocation, the sets π_i are pairwise disjoint and $\sum_{i \in A} |\pi_i| = p$, which implies that $|\bigcup_{i \in A} \pi_i| = p$. Thus, the set of labels assigned to vertices in $\bigcup_{i \in A} \pi_i$ forms a subset of $[m]$ of size p. By the definition of an (m, p)-perfect hash family, there exists a function that is injective on this set, and this function induces a partition of $V(G)$ satisfying the desired property. Let $\mathcal{S}$ be a partition of $[p]$ such that $c \in [p]$ belongs to S_i if and only if $C_c \cap \pi_i \neq \emptyset$. Finally, for every agent $i \in A$, $c \in [p]$ and $v \in \pi \cap C_c$, define, $\Lambda(i, c) = \overline{\log}_q(u_i(\pi_i)) - \overline{\log}_q(u_i(v_c))$ if it is less than t and $u_i(v_c) > 0$, and $\Lambda(i, c) = t$ otherwise. The tuple $(\mathcal{C}, \mathcal{S}, \Lambda)$ is a bucket profile, since for any pair of distinct agents $i, j \in A$, we have:

$$\sum_{\substack{c\in S_j\\ \Lambda(i,c)\neq t}} q^{-\Lambda(i,c)} \leq \sum_{\substack{c\in S_j\\ v\in\pi\cap C_c}} q^{-\lceil\log_q(u_i(\pi_i))\rceil+\lceil\log_q(u_i(v_c))\rceil} \leq q\cdot\sum_{v\in\pi_j}\frac{u_i(v)}{u_i(\pi_i)} \leq q$$

Finally, since Π respects the bucket profile $(\mathcal{C},\mathcal{S},\Lambda)$ by definition, the lemma follows. ◁

Let $(\mathcal{C},\mathcal{S},\Lambda)$ be the bucket profile in $\mathcal{B}$ that is respected by Π, whose existence is guaranteed by Claim 4.11. Let $\bar{\mu}^o$ denote the target vector attained by Π, where $\mu_i^o := \lceil\log_q(u_i(\pi_i))\rceil$ for each $i\in A$. We claim that, while the algorithm processes the bucket profile $(\mathcal{C},\mathcal{S},\Lambda)$, it never considers any target vector $\bar{\mu}$ for which $\mu_i<\mu_i^o$ for some $i\in A$. Suppose, for contradiction, that it does. Let $\bar{\mu}^\alpha$ be the first such target vector considered by the algorithm and $\bar{\mu}^\beta$ be the target vector of the previous iteration. Let $i\in A$ be the agent returned by the subroutine while considering $(\mathcal{C},\mathcal{S},\Lambda)$ and $\bar{\mu}^\beta$. By definition of $\bar{\mu}^\alpha$, we have that $\mu_i^\alpha=\mu_i^o-1$ and $\mu_j^\alpha\geq\mu_j^o$ for every $j\in A$ such that $j\neq i$, which implies $\bar{\mu}^o\leq\bar{\mu}^\beta$ and $\mu_i^\beta=\mu_i^o$. But, since Π is a valid and envy-free allocation that respects $(\mathcal{C},\mathcal{S},\Lambda)$ and attains $\bar{\mu}^o$ for agent i, by the correctness of Lemma 4.8, agent i cannot be returned by the subroutine in the iteration corresponding to $(\mathcal{C},\mathcal{S},\Lambda)$ and target vector $\bar{\mu}^\beta$. Thus, we conclude that while the algorithm processes the bucket profile $(\mathcal{C},\mathcal{S},\Lambda)$, it never considers any target vector $\bar{\mu}$ for which $\mu_i<\mu_i^o$ for some $i\in A$. Observe that this suffices to prove that the algorithm returns an allocation, since the algorithm begins the iteration corresponding to $(\mathcal{C},\mathcal{S},\Lambda)$ with $\bar{\mu}=\bar{\mu}'$, and in every iteration where it does not return an allocation, it decreases μ_i for some $i\in A$. Therefore, if the algorithm never encounters a target vector $\bar{\mu}$ with $\mu_i<\mu_i^o$ for any $i\in A$, it must eventually consider $\bar{\mu}=\bar{\mu}^o$, before reaching a target vector with at least one entry equal to negative two. By the correctness of the subroutine in Lemma 4.8, an allocation must be returned at this point, since Π is a valid and envy-free allocation that respects $(\mathcal{C},\mathcal{S},\Lambda)$ and attains $\bar{\mu}^o$ for every agent i. This concludes the proof of correctness of the algorithm.

Finally, the running time bound of $(\frac{1}{\varepsilon}\log\frac{p}{\varepsilon})^{\mathcal{O}(|\mathcal{A}|p^2)}m^{\mathcal{O}(1)}$ follows from the fact that the subroutine runs in $3^p m^{\mathcal{O}(1)}$ time, together with the observation that the algorithm considers at most $e^p p^{\mathcal{O}(\log p)}\log m\cdot(p|\mathcal{A}|)^p\cdot t^{|\mathcal{A}|p^2}$ bucket profiles in $\mathcal{B}$, and that the entries of $\bar{\mu}$ can be decreased at most $\mathcal{O}(p\,m\,|\mathcal{A}|)$ times before at least one of them becomes negative. ◀

We remark that this algorithm extends to the mandatory setting with only minor modification. In particular, it suffices to strengthen the definition of a bucket profile to ensure that the partition $\mathcal{S}$ only contains non-empty sets. The correctness of the algorithm then follows with straightforward changes to account for this additional constraint.

Finally, we conclude the section by restating Lemma 4.4 and describing the algorithm it refers to.

▶ **Lemma 4.4.** *There exists an algorithm that, given a graph G, a weight function $w: V(G)\to\mathbb{N}_{\geq 0}$, and a partition $\mathcal{C}=\{C_1,\ldots,C_k\}$ of $V(G)$ into k parts, runs in time $3^k\cdot|V(G)|^{\mathcal{O}(1)}$ and outputs a connected subgraph H of G that maximizes $\sum_{v\in V(H)}w(v)$ subject to the constraint that $|V(H)\cap C_i|=1$ for every $i\in[k]$, if such a subgraph exists, and correctly reports that no such subgraph exists otherwise.*

Proof. Let H be a subgraph of G. We use the term *weight of H* to mean the quantity $\sum_{v\in V(H)}w(v)$. We describe a dynamic programming algorithm that computes the weight of the maximum-weight connected subgraph H of G that satisfies $|V(H)\cap C_i|=1$ for every $i\in[k]$, if such a subgraph exists. Using standard backtracking techniques or self-reduction, a subgraph that realizes this value can be recovered with, in the worst case, an additional overhead of $n^{\mathcal{O}(1)}$ in the running time.

First, let us describe a function $f : V(G) \times 2^{[k]} \to \mathbb{N}_{\geq 0}$, where $2^{[k]}$ denotes the set of all subsets of $[k]$.

$$f(v,S) = \begin{cases} w(v), & \text{if } S = \{i\} \text{ and } v \in C_i, \\ -\infty, & \text{if } v \notin \bigcup_{i \in S} C_i, \\ \max\limits_{\substack{\emptyset \subsetneq S_1 \subsetneq S \\ u \in N[v]}} \{f(u,S_1) + f(v, S \setminus S_1)\}, & \text{otherwise.} \end{cases} \tag{2}$$

It is straightforward to observe that $f(v,S)$ computes the weight of the maximum-weight connected subgraph H of $G[\bigcup_{i \in S} C_i]$ that contains the vertex v. The algorithm constructs a dynamic programming table $F[v,S]$, where $v \in V(G)$ and $S \subseteq [k]$, by filling entries in increasing order of $|S|$ starting with singleton sets. The entries corresponding to larger sets, are computed using the recurrence of function f in Definition (2), after replacing recursive calls to f with lookups to the table F at the corresponding indices. After populating the table states, the algorithm checks the value of $\max_{v \in V(G)} F[v,[k]]$. If it is $-\infty$ the algorithm reports that no such subgraph exists, otherwise it returns the value. The correctness of the algorithm follows from an induction on the size of the subset S in the definition of the table entries. Finally, we observe that the running time is upper bounded by $\sum_{\substack{v \in V(G) \\ S \subseteq [p]}} 2^{|S|} \cdot |V(G)| \leq \sum_{v \in V(G)} \sum_{i \in [k]} \binom{k}{i} \cdot 2^i \cdot 1^{k-i} \cdot |V(G)| = \sum_{v \in V(G)} 3^k \cdot |V(G)| = 3^k |V(G)|^2$, which concludes the proof of the lemma. ◀

5 Concluding Remarks and Discussion

We prove that EF-ICFDis W[1]-hard parameterized by $p + |A|$, even when the input numbers are in unary representation and provide an EPAS parameterized by $p + |\mathcal{A}|$ for the problem even when input is in binary representation. We remark that our algorithm outperforms the brute force algorithm running in $\mathcal{O}(m^p \cdot n^p)$ when $p < \mathcal{O}(\log m + \log n)$. This regime remains practically relevant, particularly in scenarios involving the allocation of expensive or limited goods.

Beyond envy-freeness, other fairness notions have been explored in the incomplete and connected setting. Gahlawat and Zehavi [21] investigated three such notions, including EF1 and EFX, which relax envy-freeness by requiring that envy disappears once the most (respectively, least) valuable item is removed from every other agent's bundle. They showed that finding an EF1 (EFX) allocation parameterized by $p + |A|$ is W[1]-hard when the input is given in binary representation. Our results may provide a starting point for answering related questions to these relaxed fairness concepts.

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014*, pages 1–12, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-44777-2_1.

2 Hannaneh Akrami, Bhaskar Ray Chaudhury, Martin Hoefer, Kurt Mehlhorn, Marco Schmalhofer, Golnoosh Shahkarami, Giovanna Varricchio, Quentin Vermande, and Ernest van Wijland. Maximizing nash social welfare in 2-value instances. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):4760–4767, June 2022. doi:10.1609/aaai.v36i5.20402.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

4 Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artificial Intelligence*, 322:103965, 2023. doi:10.1016/j.artint.2023.103965.

5 Haris Aziz, Ioannis Caragiannis, Ayumi Igarashi, and Toby Walsh. Fair allocation of indivisible goods and chores. *Auton. Agents Multi Agent Syst.*, 36(1):3, 2022. doi:10.1007/s10458-021-09532-8.

6 Xiaohui Bei, Ayumi Igarashi, Xinhang Lu, and Warut Suksompong. The price of connectivity in fair division. *SIAM journal on Discrete Mathematics*, 36(2):1156–1186, 2022. doi:10.1137/20m1388310.

7 Aurélie Beynier, Yann Chevaleyre, Laurent Gourvès, Ararat Harutyunyan, Julien Lesca, Nicolas Maudet, and Anaëlle Wilczynski. Local envy-freeness in house allocation problems. *Autonomous Agents and Multi-Agent Systems*, 33:591–627, 2019. doi:10.1007/s10458-019-09417-x.

8 Vittorio Bilò, Ioannis Caragiannis, Michele Flammini, Ayumi Igarashi, Gianpiero Monaco, Dominik Peters, Cosimo Vinci, and William S Zwicker. Almost envy-free allocations with connected bundles. *Games and Economic Behavior*, 131:197–221, 2022. doi:10.1016/j.geb.2021.11.006.

9 Arpita Biswas and Siddharth Barman. Fair division under cardinality constraints. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, IJCAI'18, pages 91–97. ijcai.org, 2018. doi:10.24963/ijcai.2018/13.

10 Bernhard Bliem, Robert Bredereck, and Rolf Niedermeier. Complexity of efficient and envy-free resource allocation: Few agents, resources, or utility levels. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, IJCAI'16, pages 102–108. AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/022.

11 Sylvain Bouveret, Katarína Cechlárová, Edith Elkind, Ayumi Igarashi, and Dominik Peters. Fair division of a graph. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, IJCAI'17, pages 135–141. ijcai.org, 2017. doi:10.24963/ijcai.2017/20.

12 Robert Bredereck, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation using parametric integer linear programming. In *ECAI 2023*, pages 303–310. IOS Press, 2023. doi:10.3233/FAIA230284.

13 Robert Bredereck, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation: Lenstra empowered by n-fold integer programming. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, EC '19, pages 505–523, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3328526.3329649.

14 Robert Bredereck, Andrzej Kaczmarczyk, and Rolf Niedermeier. Envy-free allocations respecting social networks. *Artificial Intelligence*, 305:103664, 2022. doi:10.1016/j.artint.2022.103664.

15 Eric Budish. The combinatorial assignment problem: approximate competitive equilibrium from equal incomes. In *Proceedings of the Behavioral and Quantitative Game Theory: Conference on Future Directions*, BQGT '10, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1807406.1807480.

16 Ioannis Caragiannis, Evi Micha, and Nisarg Shah. A little charity guarantees fair connected graph partitioning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4908–4916, 2022. doi:10.1609/aaai.v36i5.20420.

17 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015. doi:10.1007/978-3-319-21275-3.

18 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Thekla Hamm, and Sebastian Ordyniak. The parameterized complexity of connected fair division. In Zhi-Hua Zhou, editor, *Proceedings*

of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pages 139–145. International Joint Conferences on Artificial Intelligence Organization, August 2021. Main Track. doi:10.24963/ijcai.2021/20.

19 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 5th edition, 2012.

20 Uriel Feige, Ariel Sapir, and Laliv Tauber. A tight negative example for mms fair allocations. In *International Conference on Web and Internet Economics*, pages 355–372. Springer, 2021. doi:10.1007/978-3-030-94676-0_20.

21 Harmender Gahlawat and Meirav Zehavi. Parameterized Complexity of Incomplete Connected Fair Division. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2023)*, volume 284 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2023.14.

22 Jiarui Gan, Warut Suksompong, and Alexandros A Voudouris. Envy-freeness in house allocation problems. *Mathematical Social Sciences*, 101:104–106, 2019. doi:10.1016/j.mathsocsci.2019.07.005.

23 Jugal Garg and Setareh Taki. An improved approximation algorithm for maximin shares. In *Proceedings of the 21st ACM Conference on Economics and Computation*, EC '20, pages 379–380, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3391403.3399526.

24 Jonathan Goldman and Ariel D. Procaccia. Spliddit: unleashing fair division algorithms. *SIGecom Exch.*, 13(2):41–46, January 2014. doi:10.1145/2728732.2728738.

25 Gianluigi Greco and Francesco Scarcello. The complexity of computing maximin share allocations on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2006–2013, 2020. doi:10.1609/aaai.v34i02.5572.

26 Ayumi Igarashi and Dominik Peters. Pareto-optimal allocation of indivisible goods with connectivity constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 2045–2052, 2019. doi:10.1609/aaai.v33i01.33012045.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

28 Vincent Lacroix, Cristina G Fernandes, and Marie-France Sagot. Motif search in graphs: application to metabolic networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(4):360–368, 2006. doi:10.1109/TCBB.2006.55.

29 Richard J Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 125–131, 2004. doi:10.1145/988772.988792.

30 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, USA, 23-25 October 1995*, FOCS '95, pages 182–191, USA, 1995. IEEE Computer Society. doi:10.1109/SFCS.1995.492475.

31 Trung Thanh Nguyen and Jörg Rothe. Fair and efficient allocation with few agent types, few item types, or small value levels. *Artificial Intelligence*, 314:103820, 2023. doi:10.1016/j.artint.2022.103820.

32 Benjamin Plaut and Tim Roughgarden. Almost envy-freeness with general valuations. *SIAM Journal on Discrete Mathematics*, 34(2):1039–1068, 2020. doi:10.1137/19M124397X.

33 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.

34 Warut Suksompong. Constraints in fair division. *SIGecom Exch.*, 19(2):46–61, December 2021. doi:10.1145/3505156.3505162.

Approximating Optimal Broadcast of Files in a Hose-Model Network

Thomas Erlebach
Department of Computer Science, Durham University, UK

Naveen Garg
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Sukriti Gupta
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Amitabh Trehan
Department of Computer Science, Durham University, UK

Abstract

The paper considers the problem of file sharing among peers who are connected to a common core network through links of differing upload and download capacities, as is the case in networks provisioned according to the hose model. The file is assumed to be divided into equal-sized chunks, and a peer can start sending a "chunk" of the file to another peer only after it has received the entire chunk. The objective is to share a chunk, initially residing on one of the peers, with all other peers in the least time possible. Peers can simultaneously send/receive parts of a chunk to/from multiple peers, subject to the upload and download capacity constraints. We only consider the problem of broadcasting one chunk to all peers.

We consider two different models – in the migratory model, a peer can receive the chunk from multiple peers, while in the non-migratory model, any peer can receive the chunk only from one peer. For the migratory model, introduced in this paper, we show a novel integer program and use the optimum solution to the LP-relaxation to give a schedule with makespan $e^{1/e}\texttt{OPT} + P$ where P is the time required by the slowest peer to download the chunk.

Minimising makespan in the non-migratory model is known to be NP-hard. We give a solution with makespan $18\texttt{OPT} + P$ and this is the first approximation algorithm for heterogeneous and asymmetric upload/download capacities. We also consider 2 special cases. For uniform download capacities, we obtain a solution with makespan $2\texttt{OPT}$ extending a result due to Liu [12]. For uniform upload capacities, we give the first approximation algorithm, producing makespan at most $2\texttt{OPT} + 2P$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases File sharing, scheduling, peer-to-peer networks

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.30

Funding *Naveen Garg*: Supported by Usha Hasteer Chair.

1 Introduction

We consider the problem of file sharing between peers in networks. The file to be shared initially resides with a *root* peer and has to be *broadcast*, i.e., sent to all the remaining n peers. The peers are all connected to a *core network* with links of bounded upload and download capacities. The core network is suitably provisioned so that any traffic matrix between the peers that meets the upload and download capacity constraints on each peer can be supported; this is commonly referred to as the *hose model* [6, 1]. The rate of transfer between two peers is only constrained by the upload and download capacities of the sending and receiving peer, respectively. For instance, if peer i has an upload capacity of 10Mbps and peers j, k have download capacities of 6Mbps then peer i can transfer data to peers j, k at a rate of 5Mbps or to peer j at 6Mbps and peer k at 4Mbps or vice-versa. Our objective is to schedule the transfers between peers, efficiently.


© Thomas Erlebach, Naveen Garg, Sukriti Gupta, and Amitabh Trehan;
licensed under Creative Commons License CC-BY 4.0

Such a scenario would, for instance, arise when the core is a *cloud network* [2] to which a set of enterprise clients are connected who have to share a file with all when it becomes available at one. While there has been a lot of previous work on broadcasting in networks, most of it focuses on the structure of the communication network [9, 10], which in our model plays no role.

Large files are typically distributed in "chunks". A chunk is the smallest unit of the file that must be completely available to a peer before it can start sharing any part of it with another peer. A small file can be treated as a single chunk. In this paper, we shall only be concerned with distributing one chunk residing at the root peer to the remaining peers in the network. Our model permits a chunk to be split and for a peer to send parts of the chunk to multiple peers concurrently. We consider two variants of this model. In the *non-migratory setting*, a peer must receive the entire chunk from only one peer (its parent peer) while in the *migratory setting*, a peer could receive parts of the chunk from different peers. Our objective in both settings is to schedule these transfers so that the time by which all peers receive the chunk (the *makespan*) is minimised.

We remark that it is standard in networking that certain units (such as IP packets) can only be forwarded once they have been received in full, for example, because a checksum needs to be verified before one can be sure that the unit has been received correctly and can start forwarding it. Dropping this requirement for chunks in our setting, i.e., allowing parts of a chunk to be forwarded immediately, gives rise to the *fluid limit model*, in which the makespan can be minimised in polynomial time. We emphasise that most results on broadcasting treat the message to be broadcast as an atomic unit; the chunk, on the other hand can be split and reach a peer along different paths.

If upload capacities were uniform then it would be natural to distribute the chunk in order of decreasing download capacities with the peer having the largest download capacity receiving it first. Similarly, if the download capacities were uniform it would make sense to distribute the chunk in order of decreasing upload capacity and first sending it to the peer with the largest upload capacity. When upload and download capacities are identical (the symmetric setting) both these ideas yield the same order for distributing the chunk to the peers. However, when upload and download capacities are different (for a peer and across peers) there seems to be no natural preference order for distributing the chunk. For the non-migratory setting minimizing makespan is NP-hard [8] but for the migratory setting, we do not know if the problem is NP-hard.

The problem of scheduling the broadcast of multiple chunks is also of much interest but in this paper, we limit our discussion to a single chunk.

1.1 Our Results

This paper is the first to address broadcast in the hose model in its full generality, i.e., the asymmetric heterogeneous setting where the upload and download capacities of each peer can be different. Previous papers have only considered the symmetric setting when the upload and download capacity of each peer are identical [8]; see Section 1.2 for details. The problem of finding a constant approximation for the asymmetric setting has been open, and even the setting of uniform upload capacities had not been considered earlier.

Additionally, the paper introduces the novel migratory setting that allows a peer to download from multiple peers subject to its download capacity constraint. This is a natural relaxation and does not lead to loss of data identity, as we still process the chunk as a whole. The constraint that a peer starts uploading only after receiving the complete chunk is important and separates our model from the fluid limit model.

Our first result is a novel completion time integer program for the problem of minimising makespan in the migratory setting. Formulating this integer program requires proving that a certain non-trivial property is necessary and sufficient for characterising valid broadcast schedules. Relaxing the integrality constraints gives a linear program, and we exploit the structure of the optimum solution of this LP to obtain a solution with makespan at most $e^{1/e}\texttt{OPT} + P$ (Theorem 5). P is the time required for the slowest peer to download the chunk and while it is definitely less than $\texttt{OPT}$, it could be significantly smaller.

For the non-migratory setting, we do not know how to formulate a suitable integer program, but we circumvent this difficulty by restricting attention to so-called slotted schedules. We formulate an LP by restricting attention to downloads that fit into "slots"; we establish that this restriction does not lead to a significant increase in makespan. Our algorithm for rounding the optimum solution to the linear program proceeds through three stages each of which requires careful reasoning about the structure of the solution. This yields a solution of makespan $3\texttt{OPT}' + P$ where $\texttt{OPT}'$ is the minimum makespan of any slotted schedule. Since $\texttt{OPT}' \leq 6\texttt{OPT}$ (Lemma 7) we obtain a solution of makespan at most $18\texttt{OPT} + P$ (Theorem 11).

In the Appendix, we also consider 2 special cases in the non-migratory setting. When all download capacities are identical we extend the results of [12] to obtain a 2-approximation algorithm. When all upload capacities are identical we modify the download capacities of some peers and add some dummy peers to obtain a modified instance with makespan at most $2\texttt{OPT} + P$. We then find an optimum schedule for this modified instance and convert it into a schedule for the original instance with makespan at most $2\texttt{OPT} + 2P$.

In the results mentioned above, we assume that the root peer has an upload capacity at least as large as the largest download capacity. If such is not the case, we can, in $\max\{P, 1/u(0)\}$ time units ($u(0)$ is the upload capacity of the root peer) transfer the chunk to the peer with the largest upload capacity and then run our algorithms. This increases the makespan by at most $\texttt{OPT}$ as $1/u(0)$ is also a lower bound on $\texttt{OPT}$. In the non-migratory model, no peer can receive the chunk at a rate larger than the largest upload capacity, and so we can cap all download capacities at this value. For the migratory model, we show (Theorem 5) that the schedule obtained by our algorithm has makespan $P^{1/PQ}\texttt{OPT} + P$ where Q is the ratio of the largest upload capacity to the largest download capacity of the peers.

1.2 Previous Work

The problem of *broadcasting*, where the task is to disseminate a message from a source node to the remaining nodes of a network as quickly as possible, has been studied under various models (see [9] for a survey). A popular model is the telephone model where communication happens in rounds. In each round, a peer with the message can communicate it to exactly one other peer and the objective is to minimize the number of rounds. If every node can communicate with any other node, Bar-Noy et al. [5] gave a simple recursive doubling algorithm which is optimal when the number of nodes is even and an additive 1 approximation when the number of nodes is odd. When communication can happen only between adjacent nodes of a given graph, Ravi [14] defined the notion of *poise* and used it to give a $O(\frac{\log^2 n}{\log\log n})$-approximation, later improved to a $O(\frac{\log n}{\log\log n})$-approximation by Elkin and Kortsarz [7].

In the simultaneous send/receive model a peer can send the message to one peer and receive a message from another peer in the same round. Bar-Noy et al. [5] gave an optimal algorithm for broadcasting multiple messages for this model. A natural generalisation of this model is the k-port model which allows each peer to send k messages to k peers and receive k messages from k peers in a single round. An additive 1 approximation for the minimum number of rounds needed to send multiple messages was obtained in [4].

Another natural generalisation to these models is allowing for network heterogeneity. Banikazemi et al. [3] considered a setting of heterogeneous processors (peers) capable of point-to-point communication with each other. The peers could have different sending speeds/rates and the model requires that any peer take part in at most a single message transmission at any given time. For this model they proposed a fastest node first algorithm (FNF); in each iteration, the peer with the least transmission time which has the message sends it to the peer with the least transmission time which does not have the message. Liu [12] gave a 2-approximation algorithm for minimizing makespan in this model and showed that the fastest node first algorithm is optimum if sending rates are integer multiples of each other. Khuller and Kim [11] proved that the Fastest node first algorithm is a 1.5-approximation.

Mundinger et al. [13] considered the file-sharing problem and introduced the uplink-sharing model. The file is broken into multiple chunks and a peer can receive a chunk only from 1 other peer. However, a peer could simultaneously send a chunk to multiple peers. The rate of transfer is constrained by the upload capacity of the sending peer and the authors assume that concurrent uploads from one peer would share the upload capacity equally (fair-sharing assumption). Under this assumption, they prove that for identical upload capacities, a simple greedy algorithm is optimal. The authors [13] also studied the fluid limit model where the chunks are infinitesimally small.

Goetzmann et al. [8] removed the fair sharing assumption from [13] and generalised the model by introducing download capacity constraints. They consider the setting of symmetric heterogeneous capacities – equal upload and download capacities per peer that may differ among peers. Continuing with the assumption that a peer can receive a chunk from only one other peer, they prove that the problem of minimizing makespan is strongly NP-hard and give a $(1 + 2\sqrt{2})$-approximation algorithm for a single chunk and a 5-approximation algorithm for multiple chunks.

2 Preliminaries and Notations

For $a, b \in \mathbb{Z}_+$, let $[a..b]$ denote the set of integers between (and including) a and b. We use $[n]$ as shorthand for $[1..n]$ and $(a..b]$ as shorthand for $[a + 1..b]$. For $a, b \in \mathbb{R}$, $[a, b]$ denotes the set of reals between (and including) a and b, and $(a, b]$ the reals x such that $a < x \leq b$.

The peers are numbered 0 through n, and we assume that the chunk resides at peer 0, which we refer to as the root peer. We define a *unit* of time to be the minimum time needed by the peer with the highest download capacity to download the chunk. The upload/download capacity of a peer is defined as the fraction of the chunk it can upload/download in one time unit. From our definition of a time unit, it follows that the maximum download capacity of any peer is 1. Let $u : [0..n] \to \mathbb{R}^{\geq 0}$ be the upload and $d : [n] \to (0, 1]$ the download capacities. Thus, a peer j with download capacity $d(j)$ can download the entire chunk in $1/d(j)$ time units. Note that the peer may download the chunk at a rate lower than its download capacity; in this case, the time it will take is longer than $1/d(j)$. Let $P = \max_{j \in [n]} 1/d(j)$ be the maximum time a peer would take to download the chunk if it does so at the maximum permissible rate.

To allow us to model the problems as time-indexed integer programs, we discretize our schedules. For the migratory setting, we assume that all downloads and uploads begin and end at times that are integer multiples of ϵ/n. If a peer is the i^{th} peer to receive a chunk in the optimum schedule, its completion time will increase by at most $i\epsilon/n$ due to this assumption, and this increases the makespan of the optimum schedule by at most ϵ. In this setting, time-slot $t \in \mathbb{Z}^+$, will refer to the time interval $[\epsilon(t-1)/n, \epsilon t/n)$.

In the non-migratory setting, our schedules are "slotted" and all uploads, downloads begin and end at integer times. We shall argue (Lemma 7) that this assumption does not cause a significant increase in the makespan of our solution. In this setting, time-slot $t \in \mathbb{Z}^+$, will refer to the time interval $[t-1, t)$. In both settings, time $t \in \mathbb{R}^+$ will refer to a point in time, while time-slot $t \in \mathbb{Z}^+$ will refer to an interval.

Let $\gamma = \epsilon/n$ for migratory schedules and $\gamma = 1$ for non-migratory schedules. A schedule is a function $x : [0..n] \times [1..n] \times [1..T] \to [0, 1]$ where $x(i, j, t)$ is the fraction of the chunk sent from peer i to peer j in time-slot t. Further, x satisfies the following constraints.

Upload Capacity: The total upload by peer i in time-slot t, $\sum_j x(i, j, t)$ cannot exceed its upload capacity $\gamma u(i)$.

Download Capacity: The total download by peer j in time slot t, $\sum_i x(i, j, t)$, cannot exceed its download capacity $\gamma d(j)$.

Full Download before Upload: A peer i can start uploading only after it has received the entire chunk. Thus, if $x(i, j, t) > 0$, then $\sum_{t'<t} \sum_{j'} x(j', i, t') = 1$.

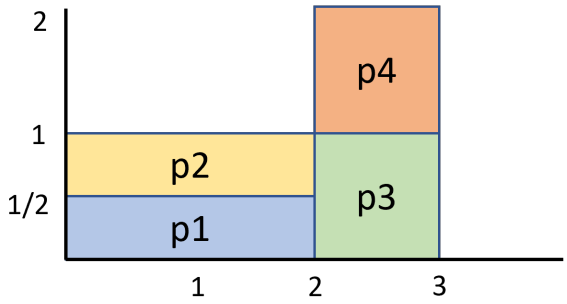


Figure 1 An example schedule: $x(0, 1, 1) = x(0, 2, 1) = x(0, 1, 2) = x(0, 2, 2) = 1/2, x(2, 3, 3) = x(0, 4, 3) = 1$. The x-axis is time and the y-axis is the total upload capacity.

Figure 1 shows a schedule for an instance with 4 peers $1, 2, 3, 4$ with (upload, download) capacities $(0, 1/2), (1, 1), (0, 1), (0, 1)$ respectively. The root peer has upload capacity 1 and sends the chunk to peers 1 and 2 during $(0, 2)$. At $t = 2$, the total upload capacity is 2 (root and peer 2 uploading), and the chunk is sent to peers 3 and 4 during $(2, 3)$. If the root peer sends the chunk to 4 and peer 2 sends the chunk to 3 during $(2, 3)$, this schedule is non-migratory, as every peer receives the chunk from a single other peer.

3 Migratory Schedules

We first consider the problem of finding a migratory schedule that minimizes makespan. In such schedules, a peer can download the chunk from multiple peers simultaneously and hence the key challenge is to ensure that there is enough upload capacity available to support all the downloads at any point in time.

3.1 Formulating a Linear Program

We formulate an integer program with completion times to determine if a single chunk can be shared by all the peers within time C. We then relax the integrality constraint to obtain a linear program that we solve to obtain a fractional solution.

3.1.1 Feasibility of download completion times

Given times $\{c_i \in \mathbb{R}^+, i \in [n]\}$, is there a schedule in which peer i completes downloading the chunk at time c_i? We renumber the peers so that $c_1 \leq c_2 \leq \cdots \leq c_n = C$. The chunk is initially with peer 0, and so $c_0 = 0$. Let $U(t)$ be the total upload capacity of all peers who have received the chunk at or before time t. Thus, for $t \in [c_{i-1}, c_i), i \in [n]$, $U(t) = \sum_{j=0}^{i-1} u(j)$, and $U(0) = u(0)$. Note that $U : [0, C] \to \mathbb{R}^+$ is a monotonically non-decreasing step-function with steps at $c_i, i \in [n-1]$; the steps in $U(t)$ are the points of discontinuity of $U(t)$. It now remains to check if there is a way of scheduling downloads such that the total download at time t does not exceed $U(t)$ and at any time the download of peer j does not exceed $d(j)$.

By time c_i the chunk has been downloaded by i peers. The total upload capacity available till time c_i, $\int_0^{c_i} U(t)dt$, must thus be at least i. This condition, although necessary, for the c_i, $i \in [n]$, to form a feasible set of download completion times, is not sufficient since we have the added constraint that at any time a peer j can only download a $d(j)$ fraction of the chunk.

Consider a time $t \in [0, T)$ and a peer j such that $t < c_j$. Note that at most a $(c_j - t)d(j)$ fraction of the chunk can be downloaded in the interval $[t, c_j]$ and if $(c_j - t)d(j) < 1$ at least a $1 - (c_j - t)d(j)$ fraction of the chunk would have to be downloaded by peer j before t. Since this is true for every peer j with $c_j > t$, it follows that all such peers download a $\sum_{j:c_j>t} \max\{0, 1 - (c_j - t)d(j)\}$ fraction of the chunk before t. Peers whose completion time is at or before t download the entire chunk before t and hence

$$\int_0^t U(t')dt' \;\geq\; |\{j : c_j \leq t\}| + \sum_{j:c_j>t} \max\{0, 1 - (c_j - t)d(j)\} \tag{1}$$

is a necessary condition for feasibility of $\{c_j, j \in [n]\}$.

3.1.2 Proving sufficiency

We next give an algorithm for scheduling downloads that establishes the sufficiency of condition 1. We pack the downloads in the upload profile U in order of increasing completion times. Let U^i denote the residual upload profile after downloads for peers 1 to i have been (see Figure 2). Thus $U^0 = U$ and since downloads for peers 1 to i are scheduled before time c_i, U^i, U are identical after time c_i. For every i, we ensure that $U^i(t)$ is a monotonically non-decreasing step-function. We now give the procedure for scheduling the download of peer j. Initially set $\alpha = c_j$ and $\beta = 1$. We wish to schedule the download of a β fraction of the chunk for peer j in $[0, \alpha]$. Let $\lambda \geq 0$ be such that $\int_0^\alpha \max\left\{U^{j-1}(t) - \lambda, 0\right\}dt = \beta$. We would like to use the part of the profile of U^{j-1} above λ to pack the download for peer j. Since U^{j-1} is monotonically non-decreasing, such a packing would be valid if and only if $U^{j-1}(\alpha) - \lambda \leq d(j)$. If this condition is true, then we have completed scheduling the download of peer j. Else, we schedule $d(j)$ units of download for peer j at every time in the interval $[\alpha', \alpha]$ where α' is the time at which U^{j-1} has a step immediately preceding the step at α. The remaining download for peer j would be scheduled in the interval $[0, \alpha']$ and this is done by repeating the above process after setting α to α' and β to $\beta - d(j)(\alpha - \alpha')$.

Note that when we repeat the above process, the value of λ decreases. Our procedure would fail if at some step we realize that the total residual upload capacity in $[0, \alpha]$ which is $\int_0^\alpha U^{j-1}(t)dt$, is less than β and thus insufficient to schedule the download of peer j. We next argue that this α is a witness to the violation of condition 1.

Let δ be the value of α at the step at which our procedure fails to schedule the download for peer j. Thus $\int_0^\delta U^{j-1}(t)dt < 1 - d(j)(c_j - \delta)$.

Algorithm 1 Scheduling download for peer j.

The function U^{j-1} gives the residual total upload capacity after the downloads for peers 1 to $j-1$ have been scheduled.
c_j is the time at which the download of peer j is to complete.
Set $\alpha \leftarrow c_j, \beta \leftarrow 1, \mu \leftarrow d(j)$.
repeat
  Find λ such that $\int_0^\alpha \max\left\{U^{j-1}(t) - \lambda, 0\right\}dt = \beta$.
  if $\lambda < 0$ **then**
    $\{c_j, j \in [n]\}$ is infeasible.
  end if
  if $U^{j-1}(\alpha) - \lambda \leq \mu$ **then**
    Let $b_j = \alpha$ and a_j be the earliest time at which $U^{j-1}(a_j) \geq \lambda$.
    Schedule $U^{j-1}(t) - \lambda$ units of download for peer j for each $t \in [a_j, b_j]$.
    $\beta \leftarrow 0$.
  else
    α' is the time at which the step of U^{j-1} ending at α begins.
    Schedule μ units of download for each $t \in [\alpha', \alpha]$.
    $\beta \leftarrow \beta - \mu(\alpha - \alpha')$, $\alpha \leftarrow \alpha'$.
  end if
until $\beta = 0$

▶ **Lemma 1.** *If the algorithm fails to schedule downloads that respect the given completion times, then*

$$\int_0^\delta U(t)dt < |\{j : c_j \leq \delta\}| + \sum_{j:c_j > \delta} \max\{0, 1 - (c_j - \delta)d(j)\}$$

Proof. To prove this lemma, we start with the following Claims.

▷ Claim 2. $\forall j \in [0..n]$, $U^j(t)$ is a monotonically non-decreasing step function. Every step in U^j is also a step in U.

Proof. We prove the statement is true by induction on j; it is true for $j = 0$ since $U^0 = U$.

From our procedure for scheduling the download of peer j it follows that there exist steps at a, b, $a \leq b \leq c_j$ in U^{j-1} such that

$$U^j(t) = \begin{cases} U^{j-1}(t) & t \in [0, a) \cup [c_j, T] \\ \lambda & t \in [a, b) \\ U^{j-1}(t) - d(j) & t \in [b, c_j) \end{cases}$$

It also follows from our choice of λ that $U^j(t) = U^{j-1}(t) < \lambda$ for $t \in [0, a)$. Monotonicity of U^j in $[b, c_j)$ follows from the fact that U^{j-1} is monotonically non-decreasing and $U^j(t) = U^{j-1}(t) - d(j)$ for $t \in [b, c_j)$. Thus it remains to argue that $U^j(b) \geq \lambda$. If $U^j(b) < \lambda$ then our choice of λ in the penultimate step of our process would have been such that $U^{j-1}(\alpha) = U^{j-1}(b) < \lambda + d(j)$ and so our procedure would have terminated at the penultimate step.

From the above argument it follows that all steps in U^{j-1} in the intervals $[0, a]$ and $[b, c_j]$ are retained in U^j. However the steps in U^{j-1} in (a, b) are not steps anymore in U^j since $U^j(t) = \lambda$ for $t \in [a, b)$. This implies all steps in U^j are also steps in U^{j-1} and hence in U. ◁

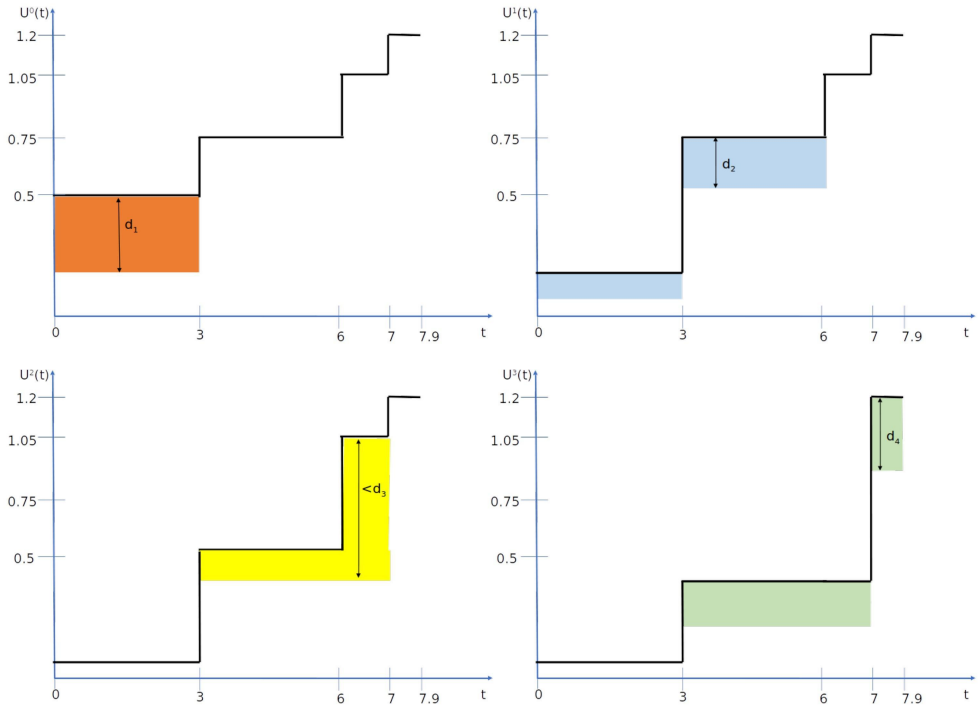


Figure 2 The upload capacities are $u(0) = 0.5, u(1) = 0.25, u(2) = 0.3, u(3) = 0.15$ and the download capacities are $d(1) = 1/3, d(2) = 0.22, d(3) = 1, d(4) = 0.3$. The figure shows the scheduling of downloads of peers 1,2,3, and 4.

Since δ is a step in U^{j-1}, the above claim implies that there exists $k \in [j-1]$ such that $c_k = \delta$.

▷ Claim 3. If peer i, $i < j$ receives a download fraction less than $d(i)$ at time $t \geq c_k = \delta$ then peer i receives the entire chunk in $[c_k, c_i]$.

Proof. Since $\delta = c_k$ is a step in U^j it is also a step in $U^i, i \in [j-1]$. If peer i received some download before time δ, then it was downloaded to an extent less than $d(i)$ at times before and after δ. Hence $\delta \in [a_i, b_i]$ which would imply that δ is no more a step in U^{i+1} leading to a contradiction. ◁

The above claim implies that if the download of a peer $i, k+1 \leq i \leq j$ is scheduled before δ, then for all $t \in [\delta, c_i]$, peer i received a $d(i)$ fraction of the chunk. This in turn implies that $\max\{0, 1 - d(i)(c_i - \delta)\}$ fraction of the chunk is downloaded by peer i before δ. Peers 1 to k complete their entire download before δ and since the algorithm fails while scheduling download for peer j it must be the case that

$$k + \sum_{i=k+1}^{j} \max\{1 - d(i)(c_i - \delta), 0\} > \int_0^{\delta} U(t)dt$$

which completes the proof of the Lemma. ◀

Thus, constraint (1) is violated, and this establishes the sufficiency of the condition.

3.1.3 An LP relaxation

The constraint (1) gives a necessary and sufficient condition for $\{c_j, j \in [n]\}$ to be a feasible set of completion times. We use this to formulate a completion time integer program that is feasible if and only if all downloads can be completed by time C. Our first step is to round the download capacities down to the nearest power of $1/(1+\delta)$, where δ is a constant to be picked later. To simplify notation, we continue to use $d(j)$ for the rounded download capacities. We define class k as the set of peers with download capacity $1/(1+\delta)^k$; the number of classes is at most $\lceil \log_{1+\delta} P \rceil$, and we denote this by L.

Let $T = \lceil nC/\epsilon \rceil$ and $x_{jt}, j \in [n], t \in [T]$ be an indicator variable that is 1 if peer j finishes downloading the chunk in the time slot t and 0 otherwise. Clearly,

$$\forall j \in [n] \quad \sum_{t \in [T]} x_{jt} = 1 \tag{2}$$

Then $U(t) = u(0) + \sum_{j \in [n]} \sum_{t' \in [t-1]} x_{jt'} u(j)$, where $U(t)$ denotes the total upload capacity in time slot t. The constraint (1) can now be formulated as

$$\forall t' \in [T] \quad \sum_{j \in [n]} \left(\sum_{t \in [t']} x_{jt} + \sum_{t \in (t'..T]} x_{jt} \max\{1 - d(j)(t - t'), 0\} \right) \leq \sum_{t \in [t']} U(t) \tag{3}$$

Relaxing integrality constraints on the variables $\{x_{jt}, j \in [n], t \in [T]\}$ gives a linear program with $n^2 C/\epsilon$ variables and $n + nC/\epsilon$ constraints. In Theorem 5 we argue that $C \leq n^2$ and therefore the LP can be solved in time polynomial in $n, 1/\epsilon$.

Figure 3 shows an instance with an integrality gap of 3/2. Peers $1, 2, 3$ have (upload, download) capacities (0,1/2),(2,1),(0,1) respectively, and the root peer has upload capacity 1. The linear program is feasible for $C = 2$ while the best integral solution has makespan 3.

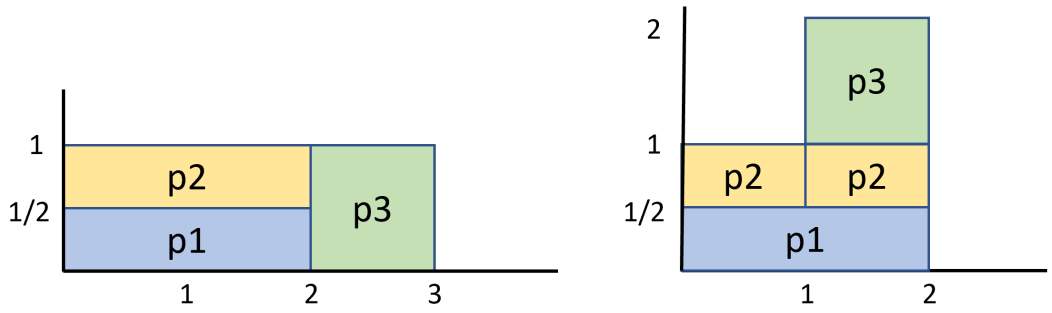


Figure 3 The LP solution on the right has $x_{1,2} = 1, x_{2,1} = 1/2, x_{2,2} = 1/2, x_{3,2} = 1$ and a makespan of 2. The best integral solution is on the left and has a makespan of 3. In both figures, the x-axis is time and the y-axis is the total upload capacity.

3.2 Constructing an integer solution

Let x be an optimum solution to the linear program and let $U_f(t) = U(t), t \in [T]$. We can modify the algorithm from Section 3.1.2 to compute the schedule corresponding to the solution x. With each $x_{jt} > 0$ we associate a pseudo-peer $\langle j, t \rangle$ which has download capacity $x_{jt} \cdot d(j)$, upload capacity $x_{jt} \cdot u(j)$ and needs to download only a x_{jt} fraction of the chunk by time slot t. The algorithm can be used to schedule the download of the chunk to these

pseudo-peers in order of increasing t. The only modification needed is that β, the fraction of the chunk that remains to be downloaded to the pseudo-peer $\langle j,t\rangle$, is initially set to x_{jt}, and γ, the largest fraction of chunk which can be downloaded in a time-slot, is now set to $x_{jt} \cdot d(j)$.

Thus, if the upload capacity in time slot t is $U_f(t)$, we can build a schedule such that an x_{jt} fraction of the chunk finishes downloading to pseudo-peer $\langle j,t\rangle$ and hence to peer j by time slot t. Let $c(j)$ be the earliest time slot such that $\sum_{t\in[c(j)]} x_{jt} = 1$. Peer j completes downloading the chunk at time slot $c(j)$ in this schedule we have built and hence it can start uploading the chunk only after time slot $c(j)$. Thus the true (or integral) upload capacity in time slot t in this schedule is $U_i(t) = u(0) + \sum_{j:c(j)<t} u(j)$. Note that $U_f(t)$, the fractional upload capacity at time slot t, is the sum of the upload capacities of all peers weighted by the fraction of the chunk that they have received by time slot t. It follows therefore, that for all time slots t, $U_f(t) \geq U_i(t)$. A feasible schedule can, in any time slot t, only permit downloads of total capacity at most the true upload capacity $U_i(t)$ and hence the schedule obtained from solution x is not feasible. In order to build a feasible schedule we need a better understanding of the difference between U_f and U_i. Informally, the following lemma shows that for two peers in the same class, the download of the peer with higher upload capacity precedes the download of the peer with lower upload capacity.

▶ **Lemma 4.** *There exists an optimum fractional solution in which* $\forall j_1, j_2$ *such that* $d(j_1) = d(j_2)$*, if* $u(j_1) > u(j_2)$ *then* $\forall t < c(j_1), x_{j_2 t} = 0$

Proof. For contradiction assume that $\exists j_1, j_2, d(j_1) = d(j_2), u(j_1) > u(j_2)$ and time slots $t_1, t_2, t_1 < t_2$ such that $x_{j_2 t_1}, x_{j_1 t_2} > 0$. Let $\delta = \min(x_{j_2 t_1}, x_{j_1 t_2})$ and we construct a new LP solution x' which is identical to x with the exception that $x'_{j_2 t_1} = x_{j_2 t_1} - \delta, x'_{j_1 t_1} = x_{j_1 t_1} + \delta$ and $x'_{j_1 t_2} = x_{j_1 t_2} - \delta, x'_{j_2 t_2} = x_{j_2 t_2} + \delta$.

We next argue that x' is also a feasible solution to the linear program. Replacing the download of peer j_2 that finishes in time slot t_1 with an equal download of peer j_1 does not change the left-hand side of inequality (3) since $d(j_1) = d(j_2)$. The same holds for replacing the download of peer j_1 ending in time slot t_2 with a download for peer j_2. Doing these exchanges together ensures that the constraint (2) is also met.

Let $U'(t) = \sum_{t'\in[t-1]} \sum_{j\in[n]} x'_{jt'} u(j)$. Since $u(j_1) > u(j_2)$, exchanging downloads of j_2 and j_1 in time slots t_1, t_2 will only increase the upload capacity at time slots $t \in (t_1..t_2]$. Thus $U'(t) > U(t), t \in (t_1..t_2]$ and they are equal otherwise. This implies that the right-hand-side of inequality 3 is only larger in x' than in x and hence the constraint is met. ◀

3.2.1 Shifting downloads within a class

Let $j_1, j_2, \ldots, j_p$ be peers of a class l such that $u(j_1) \geq u(j_2) \geq \cdots \geq u(j_p)$. From the above lemma, it follows that there exists an LP solution, say x, such that $c(j_1) \leq c(j_2) \leq \cdots \leq c(j_p)$. In time slot $t, t \in (c(j_i)..c(j_{i+1})]$, the total contribution (in solution x) by peers of class l to the true upload capacity is exactly $\sum_{h=1}^{i} u(j_h)$ while their contribution to the fractional upload capacity is at most $\sum_{h=1}^{i+1} u(j_h)$.

We now build a solution $x^{(1)}$ by shifting the downloads of peers within each class. This is done by downloading the chunk for peer j_{i+1} in the time slots in which the download for peer j_i was scheduled in the solution x. This can be done since the restriction on the fraction of the chunk that can be downloaded in a time-slot (the download capacity) is identical for peers j_i, j_{i+1}. Thus for all t and $1 \leq i \leq p$, we set, $x^{(1)}_{j_{i+1},t} = x_{j_i,t}$, where $j_1, j_2, \ldots, j_p$ are peers of a class. For now, we will assume that peer j_1 has already received the chunk from the root peer and shall remove this assumption later.

We repeat this process of shifting downloads of peers for all classes. Once this shifting is done, consider $t \in (c(j_i)..c(j_{i+1})]$ and the true upload capacity in solution $x^{(1)}$ at time slot t. The total contribution by peers of class l to this quantity is $\sum_{h=1}^{i+1} u(j_h)$, which is at least as large as their contribution to the fractional upload capacity in time slot t in solution x. Since the fractional upload capacity was sufficient to schedule all downloads, it follows that the solution $x^{(1)}$ is indeed a feasible solution. We note our assumption that the peers with the highest upload capacity in each class (the *leaders*) have already received the chunk at time $t = 0$ from the root peer and hence their upload capacity is being included in the true upload capacity available in each time slot.

3.2.2 Distributing chunks to class-leaders

We now show how to distribute the chunk to the leaders. The root peer simultaneously uploads the chunk to each of the L class leaders at a rate $1/P$, and hence all class leaders receive the chunk at time P. This requires that the root peer has an upload capacity of at least $L/P = P^{-1}\lceil \log_{1+\delta} P \rceil$.

We assume ϵ is chosen such that nP/ϵ is an integer. We construct a new solution $x^{(2)}$ by shifting the solution $x^{(1)}$ forward in time by P units. Formally, for peers $j \in [n]$ and time slots $t \in [0, T]$ we set $x^{(2)}_{j,t+nP/\epsilon} \leftarrow x^{(1)}_{j,t}$. This shifting creates nP/ϵ empty time-slots at the start, which we use for downloading the chunk from the root to all class leaders. The makespan of this modified solution is at most $C + P$.

We are now ready to prove the following theorem.

▶ **Theorem 5.** *In the migratory setting, there is a polynomial time algorithm to compute a schedule of transfers such that a root peer with upload capacity Q can share a chunk with all peers in time at most $P^{1/PQ}\mathtt{OPT} + P$, where P is the time required by the slowest peer to download the chunk. When $Q \geq 1$, our schedule has makespan $e^{1/e}\mathtt{OPT} + P$.*

Proof. Suppose $P \geq n/Q$. All peers have download capacity at least $1/P$. Since $1/P \leq Q/n$, they can trivially download the chunk simultaneously from the root peer (which has upload capacity Q) in time P, and this is the optimum. Hence, we assume $P < n/Q$. All peers can also download the chunk sequentially from the root peer in time at most $nP \leq n^2/Q$, and this is an upper bound on the makespan C.

Let I be the original instance and I' the instance with download capacities rounded down to a power of $1/(1+\delta)$.

▷ Claim 6. $\mathtt{OPT}(I') \leq (1+\delta)\mathtt{OPT}(I)$.

Let C' be a guess on the makespan of the instance I'. Note that C' is at least P and at most nP. By doing a binary search on C' and solving the linear program for the instance I' on each guess, we determine the smallest value of C', say C^*, for which the linear program is feasible. Note that $C^* \leq \mathtt{OPT}(I')$ and let x be the solution to the linear program. We modify x as discussed in previous sections to obtain a feasible solution $x^{(2)}$ with makespan $C^* + P \leq \mathtt{OPT}(I') + P \leq (1+\delta)\mathtt{OPT}(I) + P$.

The upload capacity of the root peer Q, must be at least $P^{-1}\log_{1+\delta} P$ and this implies $\delta \geq P^{1/PQ} - 1$. With this choice of δ, the schedule returned by our algorithm has makespan $P^{1/PQ}\mathtt{OPT} + P$. When $Q \geq 1$, the makespan is at most $e^{1/e}\mathtt{OPT} + P$. ◀

If the peer with the largest upload capacity is different from the root then we incur an additional $\max\{1/u(0), P\}$ time in transferring the chunk from the root to this peer.

4 The non-migratory setting

This section considers the setting when a peer can download the chunk from only one peer – the *parent peer*. Let T be an estimate of the minimum makespan of a slotted schedule, defined below. We divide the interval $[0, T]$ into slots of different sizes. Let $K = \lfloor \log_2 T \rfloor$. For $k \in [0..K]$, a *class-k-slot* is the time-interval $[p \cdot 2^k, (p+1) \cdot 2^k), p \in [0..\lfloor T/2^k \rfloor)$ and we refer to this interval as the slot (k, p). For a slot (k, p) and time t, we let "$t < (k, p)$" denote $t < p \cdot 2^k$, "$t > (k, p)$" denote $t \geq (p+1) \cdot 2^k$, and "$t \in (k, p)$" denote $t \in [p \cdot 2^k, (p+1) \cdot 2^k)$. Note that any two slots – of the same or different classes – are either disjoint or one slot contains the other.

4.1 Slotted Schedules

A peer j is in class l if $1/2^l \leq d(j) < 1/2^{l-1}$; let $\phi(j)$ denote the class of peer j. A peer downloads the chunk in slot (k, p) if the chunk is downloaded in the interval $[p \cdot 2^k, (p+1) \cdot 2^k)$ at a uniform rate $1/2^k$. A peer j can download the chunk in a slot of class k only if $k \geq \phi(j)$. We call a schedule *slotted* if every peer downloads the chunk in some slot and we formulate an LP to minimize the makespan of a slotted schedule. The following Lemma, whose proof we omit due to space constraints, argues that the makespan of a slotted schedule is not significantly more than the makespan of the optimum schedule.

▶ **Lemma 7.** *Any non-migratory schedule S with makespan T can be converted into a non-migratory slotted schedule S' with makespan at most $6T$.*

4.2 An LP for slotted schedules

Let T be our estimate of the minimum makespan and let $x_{ijkp}, i \in [0..n], j \in [n], k \in [0..K], p \in [0..\lfloor T/2^k \rfloor)$ be 1 if peer j downloads the chunk from peer i in slot (k, p). Note that peer i can upload the chunk in a slot (k, p) only if $2^{-k} \leq u(i)$. We combine this with our earlier observation that $k \geq \phi(j)$ to define variables x_{ijkp} only for $k \geq \kappa_{ij}$ where $\kappa_{ij} = \max(\phi(j), \log_2 u(i)^{-1})$

1. The constraint $\sum_{k,p} \sum_{i \in [0..n]} x_{ijkp} \geq 1$ ensures that peer j downloads the chunk in some slot.
2. Let y_{jt} denote the fraction of the chunk downloaded by peer j before time t. Note that this is at most $\sum_{k,p:(k,p)<t} \sum_{i \in [0..n]} x_{ijkp}$. Peer 0 (the root) has the chunk available to it at time 0 and hence $y_{0t} = 1, \forall t \in [T]$.
3. The total downloads from peer i assigned to slots containing t is $\sum_{k,p:t \in (k,p)} \sum_{j \in [n]} x_{ijkp} \cdot 2^{-k}$ and this cannot exceed $u(i) y_{it}$ which is the total upload capacity of peer i at time t.

This then yields the following system of linear inequalities.

$$\sum_{k,p} \sum_{i \in [0..n]} x_{ijkp} \geq 1 \qquad j \in [n] \qquad (4)$$

$$\sum_{k,p:(k,p)<t} \sum_{i \in [0..n]} x_{ijkp} \geq y_{jt} \qquad j \in [n], t \in [T] \qquad (5)$$

$$\sum_{k,p:t \in (k,p)} \sum_{j \in [n]} x_{ijkp} \cdot 2^{-k} \leq u(i) y_{it} \qquad i \in [0..n], t \in [T] \qquad (6)$$

$$x_{ijkp} \in [0, 1] \qquad i \in [0..n], j \in [n], k \in [\kappa_{ij}..K], p \in [0..\lfloor T/2^k \rfloor)$$

$$y_{jt} \in [0, 1] \qquad t \in [T]$$

$$y_{0t} = 1 \qquad t \in [T]$$

If the above linear system is infeasible, our estimate of T is too low and we double T. By performing a binary search on T we determine the smallest T, say T^*, such that the linear system is feasible and let x be this feasible solution.

4.3 Rounding to a non-migratory schedule

An important property of x, along the lines of Lemma 4, is that if j_1, j_2 are peers of the same class and $u(j_1) > u(j_2)$, then j_2 cannot be downloading the chunk in a slot that ends strictly before any slot in which j_1 downloads the chunk.

▶ **Lemma 8.** *Let $j_1, j_2 \in \phi^{-1}(l)$ be such that $u(j_1) > u(j_2)$. Let $x_{ij_1kp} > 0$ and $t = (p+1)2^k$. Then for all i', k', p' such that $(k', p') < t$ and $i' \in [n]$, $x_{i'j_2k'p'} = 0$.*

Thus for each class, the chunk is received by peers of this class in decreasing order of their upload capacity. For each class l order the peers of this class by decreasing upload capacity, breaking ties arbitrarily and let $\langle l, i\rangle$ be the i^{th} peer in this ordering; it would be convenient to define $\langle l, 0\rangle$ as 0 (the root peer). Let $y_{lkp} = \sum_{i\in[n]} \sum_{j\in\phi^{-1}(l)} x_{ijkp}$ be the total extent to which peers of class l download the chunk in a slot (k, p). To determine which slot each peer of class l downloads the chunk in, we order slots (k, p) by increasing end-time. Then peer $\langle l, 1\rangle$ downloads the chunk in the earliest slots of this ordering for which the sum of y_{lkp} is 1. In general, if $c(l, i)$ denotes the time at which the chunk is fully downloaded by peer $\langle l, i\rangle$ then $c(l, i) = t$ is the earliest time at which $\sum_{k,p:(k,p)\le t} y_{lkp} \ge i$. We define $c(l, 0)$ as 0. Thus, $0 = c(l, 0) \le c(l, 1) \le c(l, 2) \le \cdots \le c(l, i) \le \cdots$. Note that a peer could be downloading the chunk from many different peers and hence, x, y is a migratory schedule. We now modify x, y in a sequence of steps to obtain a non-migratory schedule.

4.3.1 Ensuring a peer uploads only after downloading the entire chunk

From the preceding discussion it follows that for $j = \langle l, h\rangle$, $y_{jt} = 0$ for $t < c(l, h-1)$ and $y_{jt} = 1$ for $t \ge c(l, h)$. Note that y_{jt} is monotonically non-decreasing with time t and increases from 0 to 1 in the interval $[c(l, h-1), c(l, h)]$. In the linear system, a y_{jt} fraction of the upload capacity of peer j is available at time t and thus in the solution x, peer j can start uploading the chunk after time $c(l, h-1)$ and before time $c(l, h)$. However, recall that in a valid schedule, peer j can upload the chunk to the other peers only after it has completed the download at time $c(l, h)$. We resolve this by

1. assuming that at time 0, the chunk is available to the first peer of each class, and
2. scheduling the downloads by peer $\langle l, h\rangle$, $h \ge 2$, in slots which peer $\langle l, h-1\rangle$ uses to download the chunk.

Peer $\langle l, h\rangle$ would then complete its download by time $c(l, h-1)$ and the uploads by peer $\langle l, h\rangle$ after time $c(l, h-1)$ in the modified solution would now be valid. The modified solution $(x^{(1)}, y^{(1)})$ is defined by

$$
\begin{aligned}
x^{(1)}_{i\langle l,h\rangle kp} &= x_{i\langle l,h-1\rangle kp} && i \in [0..n], k \in [\kappa_{ij}..K], p \in [0..\lfloor T/2^k \rfloor], l \in [0..L], h \ge 2 \\
y^{(1)}_{\langle l,h\rangle t} &= 1 && t \in [c(l,h-1)..T], l \in [0..L], h \ge 1 \\
y^{(1)}_{\langle l,h\rangle t} &= 0 && t \in [0..c(l,h-1)-1], l \in [0..L], h \ge 2 \\
y^{(1)}_{0t} &= 1 && t \in [0..T]
\end{aligned}
$$

Let $Z \subseteq [n]$ be the peers who are not class leaders. We next argue that the solution $(x^{(1)}, y^{(1)})$ satisfies the constraints of our linear system for peers in Z.

▶ **Lemma 9.** *The solution $(x^{(1)}, y^{(1)})$ satisfies constraint (6) and constraints (4) and (5) for peers in Z.*

Proof. Consider a peer $\langle l, h\rangle, h > 1$. Since $x^{(1)}_{i\langle l,h\rangle kp} = x_{i\langle l,h-1\rangle kp}$, we have

$$\sum_{k,p} \sum_{i\in[0..n]} x^{(1)}_{i\langle k,h\rangle kp} = \sum_{k,p} \sum_{i\in[0..n]} x_{i\langle l,h-1\rangle kp} \geq 1$$

where the second inequality follows from the fact that x is a feasible solution and satisfies constraint (4).

When viewed as a function of t, $y^{(1)}_{\langle k,h\rangle t}$ increases from 0 to 1 at $t = c(l, h-1)$. Since $y_{\langle k,h-1\rangle t} \geq 1$ for all $t \geq c(l, h-1)$ it follows that for all $t \in [0..T]$, $y^{(1)}_{\langle k,h\rangle t} \leq y_{\langle k,h-1\rangle t}$. Thus,

$$\sum_{k,p:(k,p)<t} \sum_{i\in[0..n]} x^{(1)}_{i\langle k,h\rangle kp} = \sum_{k,p:(k,p)<t} \sum_{i\in[0..n]} x_{i\langle k,h-1\rangle kp} \geq y_{\langle k,h-1\rangle t} \geq y^{(1)}_{\langle k,h\rangle t}$$

and this establishes constraint (5)

Finally to prove constraint (6) we observe that for any peer $i \in [0..n]$ and slot (k, p), $\sum_{j\in Z} x^{(1)}_{ijkp} \leq \sum_{j\in[n]} x_{ijkp}$. Thus

$$\sum_{k,p:t\in(k,p)} \sum_{j\in Z} x^{(1)}_{ijkp} \cdot 2^{-k} \leq \sum_{k,p:t\in(k,p)} \sum_{j\in[n]} x_{ijkp} \cdot 2^{-k} \leq u(i) y_{it} \leq u(i) y^{(1)}_{it}$$

where the last inequality follows from the fact that for any peer $\langle l, h\rangle, h \geq 2$, $y^{(1)}_{\langle k,h\rangle t}$ increases from 0 to 1 at $t = c(l, h-1)$ while $y_{\langle k,h\rangle t}$ remains 0 till $c(l, h-1) - 1$ and is at most 1 at $t \geq c(l, h-1)$. For a peer $\langle l, 1\rangle$, $y^{(1)}_{\langle l,1\rangle t} = 1$ for all $t \in [0..T]$ while $y_{\langle l,1\rangle t} \leq 1$. ◀

4.3.2 Ensuring a peer downloads from only one peer

Note that in $(x^{(1)}, y^{(1)})$ a peer could be downloading the chunk from multiple peers and so $(x^{(1)}, y^{(1)})$ is a migratory solution. We will now modify this solution so that each peer downloads the chunk completely in one slot from one peer. Let $x^{(1)}_{i\cdot kp} = \sum_{j\in[n]} x^{(1)}_{ijkp}$ be the total extent to which the chunk is downloaded from peer i in slot (k, p).

We now construct a bipartite graph $G = (U, V, E)$ where U contains a vertex for each peer which is not a class leader. For each triple $(i, k, p), i \in [n], k \in [0..K], p \in [0..\lfloor T/2^k \rfloor]$ we have a vertex in V if $x^{(1)}_{i\cdot kp} > 0$. Let $b : V \to \mathbb{Z}^+$ be a function on the vertices in V defined such that for $(i, k, p) \in V$, $b(i, k, p) = \lceil x^{(1)}_{i\cdot kp} \rceil$. For peer $j = \langle l, h\rangle, h \geq 2$ we include edge $(j, (i, k, p))$ in E if $x^{(1)}_{ijkp} > 0$ and $(k, p) < c(l, h-1)$.

The graph G and the function b define an instance of an assignment problem in which we wish to assign every vertex $u \in U$ to a vertex in $v \in V$, such that $(u, v) \in E$ and the number of vertices assigned to any vertex $v \in V$ is at most $b(v)$. For integer b it is known that a fractional assignment implies the existence of an integer assignment. The variables $x^{(1)}_{ijkp}$ define a fractional assignment as follows

1. For edge $(j, (i, k, p)) \in E$, assign vertex $j \in U$ to vertex $(i, k, p) \in V$ to an extent $x^{(1)}_{ijkp}$.
2. For $j = \langle l, h\rangle, t = c(l, h-1), h \geq 2$, the feasibility of solution $(x^{(1)}, y^{(1)})$ implies

$$\sum_{k,p:(k,p)<t} \sum_{i\in[0..n]} x^{(1)}_{ijkp} \geq y_{jt} = 1.$$

   Hence vertex j is assigned to a total extent of 1 in this fractional assignment.
3. $\sum_j x^{(1)}_{ijkp} = x^{(1)}_{i\cdot kp} \leq \lceil x^{(1)}_{i\cdot kp} \rceil = b(i, k, p)$ and hence the total fractional assignment of vertices of U to a vertex $(i, k, p) \in V$ is no more than $b(i, k, p)$.

An integer assignment can be computed by finding a maximum flow in a network obtained by adding vertices s, t and edges $(s, u), u \in U$ and $(v, t), v \in V$ to the graph G. All edges of G and edges incident to s are assigned unit capacity while edges $(v, t), v \in V$ are assigned capacity $b(v)$. Since all edge capacities are integer a maximum $s - t$ flow in this network is integral and edges of G carrying unit-flow give an integer assignment. We use this assignment to define an integral solution $x^{(2)}$ by setting $x^{(2)}_{ijkp}$ to 1 iff the vertex corresponding to peer j in U is assigned to vertex $(i, k, p) \in V$. We also set $y^{(2)}_{jt} = y^{(1)}_{jt}, j \in [n], t \in [0..T]$.

▶ **Lemma 10.** *$(x^{(2)}, y^{(2)})$ satisfies constraints (4) and (5) for peers in Z. If upload capacities were three times the original then $(x^{(2)}, y^{(2)})$ also satisfies constraint (6).*

Proof. In the assignment we use to define $x^{(2)}$, every peer in Z is assigned to a (peer, slot) pair. This implies that for $j \in Z$,

$$\sum_{k,p} \sum_{i \in [0..n]} x^{(2)}_{ijkp} \geq 1$$

which is constraint (4).

For a peer $\langle l, h\rangle, h \geq 2$, we have set $y^{(2)}_{\langle l,h\rangle t} = y^{(1)}_{\langle l,h\rangle t} = 0$ for $t < c(l, h-1)$ and hence we only need to check constraint (5) for $t = c(l, h-1)$. From our construction of graph G it follows that in the integer assignment, peer $\langle l, h\rangle$ is assigned to a vertex (i, k, p) such that $(k, p) < c(l, h-1)$. This implies that for $t = c(l, h-1)$

$$\sum_{k,p:(k,p)<t} \sum_{i \in [0..n]} x^{(2)}_{i\langle k,h\rangle kp} \geq 1 \geq y^{(2)}_{\langle l,h\rangle t}$$

which implies constraint (5) is satisfied.

As concerns constraint (6), note that for any time $t \in [0..T]$ and peer $i \in [0..n]$

$$\begin{aligned}\sum_{k,p:t\in(k,p)} \sum_{j\in[n]} x^{(2)}_{ijkp} 2^{-k} &< \sum_{k,p:t\in(k,p)} x^{(1)}_{i\cdot kp} 2^{-k} + \sum_{k,p:t\in(k,p),2^{-k}\leq u(i)} 2^{-k} \\ &= \sum_{k,p:t\in(k,p)} \sum_{j\in[n]} x^{(1)}_{ijkp} 2^{-k} + \sum_{k=-\log_2 \min\{1,u(i)\}}^{K} 2^{-k} \\ &\leq u(i) y^{(1)}_{it} + 2 \cdot \min\{1, u(i)\} \leq 3u(i) y^{(2)}_{it}\end{aligned}$$

where the last inequality follows from the fact that peer i is uploading the chunk at time t in solution $(x^{(1)}, y^{(1)})$, and hence $y^{(1)}_{it} = y^{(2)}_{it} = 1$. ◀

Since every peer downloads the chunk from one peer the solution $(x^{(2)}, y^{(2)})$ is non-migratory. However, it violates the constraint that the total download scheduled from a peer at any time should not exceed the upload capacity of that peer. In addition, we need to distribute the chunk to class leaders.

4.3.3 Scaling and shifting the schedule

We first project the solution $(x^{(2)}, y^{(2)})$ to obtain a solution that does not reference slots. Let $x^{(3)}_{ijt} = \sum_{k,p:t\in(k,p)} x^{(2)}_{ijkp} 2^{-k}$ be the fraction of the chunk sent from peer i to peer j in time-slot t. For $j \in Z$, we have

$$\sum_{i\in[0..n]} \sum_{t\in[0..T]} x^{(3)}_{ijt} = \sum_{i\in[0..n]} \sum_{t\in[0..T]} \sum_{k,p:t\in(k,p)} x^{(2)}_{ijkp} 2^{-k} = \sum_{i\in[0..n]} \sum_{k,p} x^{(2)}_{ijkp} \geq 1$$

and hence every peer in Z receives the complete chunk. Since $x^{(2)}$ is 0/1, peer j downloads the chunk in some slot (k,p) where $k \geq \phi(j)$. This implies $x^{(3)}_{ijt} \leq 2^{-\phi(j)} \leq 1/d(j)$ for all $t \in [0..T]$ and hence no peer downloads at a rate more than its download capacity.

Since $y^{(2)}$ is 0/1 and

$$x^{(3)}_{ijt} = \sum_{k,p:t\in(k,p)} x^{(2)}_{ijkp} 2^{-k} \leq \sum_{j\in[n]} \sum_{k,p:t\in(k,p)} x^{(2)}_{ijkp} 2^{-k} \leq 3u(i) y^{(2)}_{it},$$

$x^{(3)}_{ijt} > 0$ implies $y^{(2)}_{it} = 1$. Hence, peer i uploads the chunk only after it has completed its download. Finally, the total upload by peer i in time slot t can be bounded as

$$\sum_{j\in[n]} x^{(3)}_{ijt} \leq \sum_{j\in[n]} \sum_{k,p:t\in(k,p)} x^{(2)}_{ijkp} 2^{-k} \leq 3u(i) y^{(2)}_{it} \leq 3u(i).$$

To ensure that this does not exceed the upload capacity of peer i, we scale and stretch the solution $x^{(3)}$ by 3. Formally, for $t \in [0..T]$, $j \in Z$ and $i \in [0..n]$ we set

$$x^{(3)}_{ij(3t)} = x^{(3)}_{ij(3t+1)} = x^{(3)}_{ij(3t+2)} = x^{(3)}_{ijt}/3$$

With this modification, the total upload by peer i in a time slot does not exceed its upload capacity $u(i)$.

Finally, we shift this solution by P time units and use the interval $[0,P]$ to distribute the chunk to class leaders as was done for the migratory setting. The final solution, which we denote by x^*, is defined as $x^*_{ij(t+P)} = x^{(3)}_{ijt}$. Note that the makespan of the solution x^* is $3T + P$ where T is the minimum makespan of a slotted schedule. By Lemma 7, $T \leq 6T^*$ where T^* is the minimum makespan of a non-migratory schedule.

▶ **Theorem 11.** *There is a polynomial time algorithm to find a non-migratory schedule of makespan at most $18T^* + P$ where T^* is the minimum makespan of a non-migratory schedule and $P \leq T^*$ is the maximum, over all peers, of the minimum time that a peer needs to download a chunk from root.*

5 Conclusion

The paper takes a linear programming approach for scheduling transfers of a chunk of a file in a hose-model network and presents the first results for the general setting when the upload and download capacities of peers could be different – for each peer and also across peers. We revisit the assumption made in earlier papers that a peer can download the chunk from only one other peer (We call this the non-migratory setting). The migratory setting has no such constraint. We present approximation results for makespan in both these settings and various special cases. The non-migratory assumption meant that the solution defined a tree and hence all previous approaches were in effect focused on constructing a suitable tree. Whether the migratory setting (like the non-migratory setting) is NP-hard needs to be studied.

Extending our approach to multiple chunks seems challenging. For sharing multiple chunks one would like to design a schedule which is periodic – the schedule for distributing each chunk is identical, albeit shifted in time. Periodic schedules are easy to implement in a network since the same upload and download policies are followed for every chunk. However, in a periodic schedule the upload capacity for a chunk is no more a monotonically non-decreasing function and this causes most of our arguments to break down.

It would be interesting to see if the algorithm in this paper can be redesigned so that it does not involve solving a linear program or if the linear program could be solved very efficiently perhaps using a combinatorial algorithm.

References

1 Satyajeet Singh Ahuja, Varun Gupta, Vinayak Dangui, Soshant Bali, Abishek Gopalan, Hao Zhong, Petr Lapukhov, Yiting Xia, and Ying Zhang. Capacity-efficient and uncertainty-resilient backbone network planning with hose. In Fernando A. Kuipers and Matthew C. Caesar, editors, *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, pages 547–559. ACM, 2021. doi:10.1145/3452296.3472918.

2 What is cloud networking? accessed on 27/01/2025. URL: https://aws.amazon.com/what-is/cloud-networking/.

3 Mohammad Banikazemi, Vijay Moorthy, and Dhabaleswar K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *1998 International Conference on Parallel Processing (ICPP '98), 10-14 August 1998, Minneapolis, Minnesota, USA, Proceedings*, pages 460–467. IEEE Computer Society, 1998. doi:10.1109/ICPP.1998.708518.

4 Amotz Bar-Noy and Ching-Tien Ho. Broadcasting multiple messages in the multiport model. *IEEE Trans. Parallel Distributed Syst.*, 10(5):500–508, 1999. doi:10.1109/71.770196.

5 Amotz Bar-Noy, Shlomo Kipnis, and Baruch Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *Discret. Appl. Math.*, 100(1-2):1–15, 2000. doi:10.1016/S0166-218X(99)00155-9.

6 Nick G. Duffield, Pawan Goyal, Albert G. Greenberg, Partho Pratim Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merwe. A flexible model for resource management in virtual private networks. In Lyman Chapin, James P. G. Sterbenz, Guru M. Parulkar, and Jonathan S. Turner, editors, *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 30 - September 3, 1999, Cambridge, Massachusetts, USA*, pages 95–108. ACM, 1999. doi:10.1145/316188.316209.

7 Michael Elkin and Guy Kortsarz. Sublogarithmic approximation for telephone multicast. *Journal of Computer and System Sciences*, 72(4):648–659, 2006. doi:10.1016/j.jcss.2005.12.002.

8 Kai-Simon Goetzmann, Tobias Harks, Max Klimm, and Konstantin Miller. Optimal file distribution in peer-to-peer networks. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 210–219. Springer, 2011. doi:10.1007/978-3-642-25591-5_23.

9 Sandra Mitchell Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. doi:10.1002/net.3230180406.

10 J. Hromkovic, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting and gossiping). In F. Hsu and D.-Z. Du, editors, *Combinatorial Network Theory*, pages 125–212. Kluwer Academic Publishers, 1995.

11 Samir Khuller and Yoo Ah Kim. Broadcasting in heterogeneous networks. *Algorithmica*, 48(1):1–21, 2007. doi:10.1007/s00453-006-1227-9.

12 Pangfeng Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002. doi:10.1006/jagm.2001.1204.

13 Jochen Mundinger, Richard R. Weber, and Gideon Weiss. Optimal scheduling of peer-to-peer file dissemination. *J. Sched.*, 11(2):105–120, 2008. doi:10.1007/s10951-007-0017-9.

14 R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time (extended abstract). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 202–213. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365693.

A Special cases of non-migratory schedules

The high approximation guarantee of Theorem 11 motivates us to study two special cases of the non-migratory model – when all peers have the same download capacity and when all peers have the same upload capacity. For uniform download capacities, we adapt a proof by Liu [12] to show that the Greedy algorithm produces the optimum makespan for instances where all upload capacities are a power of 2; the full details for this special case and proofs of Lemma marked with a † are deferred to the full paper.

A.1 Uniform upload capacity

We now assume that $u(j) = u$ for all j and recall $d(j) \in (0, 1]$. If $u < 1$ then peer i can transfer the chunk to peer j at a maximum rate $\min\{d(j), u\}$. Since a peer can receive the chunk from only one peer, there is no loss of generality in limiting download capacities to u. Hence for all j, $d(j) \in (0, u]$. Next, we scale all download and upload capacities by u to get an instance, I, with upload capacity 1 and $d(j) \in (0, 1], j \in [n]$.

We now consider a procedure for scheduling downloads of peers that would involve modifying the download capacity of some peers besides adding some dummy peers. We begin by rounding down $d(j)$ to the nearest power of $1/2$ and renumber peers so that $d(1) \geq d(2) \geq \cdots \geq d(n)$. For the rest of this section, we will assume $d(j), j \in [n]$ is a power of $1/2$. We will schedule the downloads by peers in *blocks* in the order 1 to n and let j be the earliest peer whose download has not yet been scheduled. Let t be the earliest time by which peers $1..(j-1)$ have completed their download and note that j peers (including the root) have received the chunk by time t and the upload capacity at time t, $U(t)$, is j. Initially at $t = 0$, $U(0) = j = 1$.

Let $k = U(t)/d(j)$ and note that k downloads at rate $d(j)$ can be scheduled during $(t, t + 1/d(j)]$; in subsequent discussion, we refer to this time interval as a *block*. Peers $j, j + 1, \ldots, (j + k - 1)$ would be the k peers downloading the chunk in this block and we increase their download capacities to $d(j)$. If $n < j + k - 1$, i.e., the number of remaining peers is less than k then we create $j + k - n - 1$ dummy peers with upload capacity 1 and download capacity $d(j)$. At the end of this round the value of j increases to $j + k$ while that of t increases to $t + 1/d(j)$. We continue to the next round if $j < n$. Let T_0 be the value of t when this procedure completes and let I' be the instance obtained. Note that I' might have more peers than the original instance I and in I' some peers might have a download capacity different from their download capacity in I. Let S' be the schedule obtained and note that S' is a valid schedule for I' and has makespan T_0.

▶ **Lemma 12** (†). *$OPT(I') \leq 2OPT(I) + P$.*

We now convert the schedule S' into a schedule, S, which is valid for the instance I and has makespan at most $T_0 + P$. Consider a block B in S' in which peers j to $(j + k - 1)$ download the chunk during the interval $(t, t + 1/d(j)]$. Let $j' = j + k, t' = t + 1/d(j)$. Note that B', the block following B in S', corresponds to the interval $(t', t' + 1/d(j')]$. We observe that peers who download the chunk in block B have a download capacity larger than $d(j')$ and can therefore be scheduled in block B' in a valid schedule for instance I. This suggests constructing S from S' by simply shifting downloads of peers in a block to the following block. A block, $(T_0, T_0 + P]$ is appended to S to accommodate the downloads of peers in the last block of S'. Hence S has makespan $T_0 + P$.

The last piece of the argument is to prove that $T_0 \leq OPT(I')$. To prove this we shall show that a makespan of $T_0 - 1$ is infeasible for instance I'. To show this we present a technique to argue infeasibility of a makespan value for a given instance of the problem.

Let $y : [T] \to \mathbb{R}_{\geq 0}$ be a monotonically non-increasing assignment of weights to time slots. If $U(t)$ is the upload capacity in time slot t in a given schedule x then we refer to the quantity $\sum_{t\in[T]} y(t)U(t)$ as the total weighted upload capacity of x and denote it by $W(x, y)$.

Let $y[a..b]$ denote the sum $\sum_{t\in[a..b]} y(t)$. If peer j finishes its download at time t in x then it contributes $y[t..T]u(j)$ to $W(x, y)$. The download by peer j consumes at least $y((t - d(j)^{-1})..t]d(j)$ units of $W(x, y)$. Thus the net contribution of peer j to $W(x, y)$ if it completes its download at time t in x is $y(t..T]u(j) - y((t - d(j)^{-1})..t]d(j)$ and we denote this by $w(j, t)$. Finally, for $j \in [n]$, let $w(j) = \max_{t\in[d(j)^{-1}..T]} w(j, t)$ and let $w(0)$, the net contribution of peer 0 to the $W(x, y)$, equal $u(0)y[1..T]$. Note that $w(j), j \in [0..n]$ is independent of x but depends on our choice of y.

▷ Claim 13. If T is a feasible value for the makespan then for any choice of y, $\sum_{j\in[0..n]} w(j) \geq 0$.

Proof. Let x be a schedule with makespan T and $x_{ijt}, i \in [0..n], j \in [n], t \in [T]$ be the fraction of the chunk that peer j downloads from peer i in time slot t. Let $c(j)$ be the time at which peer j completes its download. For any peer j, $\sum_{t\in[c(j)]} \sum_{i\in[0..n]} x_{ijt} = 1$ and for any time $t \in [c(j)]$, $\sum_{i\in[0..n]} x_{ijt} \leq d(j)$. This together with the fact that y is non-increasing implies that for all $j \in [n]$,

$$\sum_{t\in[c(j)]} \sum_{i\in[0..n]} x_{ijt}y(t) \geq y((c(j) - d(j)^{-1})..c(j)]d(j). \tag{7}$$

For all $t \in [T]$,

$$\sum_{i\in[0..n]} \sum_{j\in[n]} x_{ijt} \leq U(t) = \sum_{j:c(j)<t} u(j).$$

Multiplying the above inequality by $y(t)$ and summing over all $t \in [T]$ yields

$$\sum_{t\in[T]} y(t) \sum_{i\in[0..n]} \sum_{j\in[n]} x_{ijt} \leq \sum_{t\in[T]} y(t) \sum_{j:c(j)<t} u(j) = \sum_{j\in[n]} u(j) \sum_{t\in[c(j)+1..T]} y(t)$$

We now sum inequality (7) for all $j \in [n]$ and combine it with the above inequality to obtain

$$\sum_{j\in[n]} y((c(j) - d(j)^{-1})..c(j)]d(j) \leq \sum_{t\in[T]} \sum_{i\in[0..n]} \sum_{j\in[n]} x_{ijt}y(t) \leq \sum_{j\in[n]} y(c(j)..T]u(j)$$

which implies that $\sum_{j\in[n]} w(j, c(j)) \geq 0$. The claim follows from the fact that $w(j) \geq w(j, c(j)), j \in [n]$. ◁

The above claim implies that if we can find an assignment of weights $y : [T_0 - 1] \to \mathbb{R}_{\geq 0}$ such that $\sum_{j\in[0..n]} w(j) < 0$ then we can argue that $OPT(I') > T_0 - 1$.

▶ **Lemma 14** (†). $T_0 \leq OPT(I')$

Combining Lemmas 12 and 14 we obtain $T_0 \leq 2OPT(I) + P$. Hence the makespan of our schedule S is at most $T_0 + P \leq 2OPT(I) + 2P$.

▶ **Theorem 15.** *There exists a polynomial time algorithm for the broadcast problem in the hose model with uniform upload capacities that finds a solution with makespan at most* $2\mathsf{OPT} + 2P$.

Token Sliding Independent Set Reconfiguration on Block Graphs*

Mathew C. Francis
Indian Institute of Technology, Palakkad, India

Veena Prabhakaran
IITM Pravartak Technologies Foundation, Chennai, India

Abstract

Let S be an independent set of a simple undirected graph G. Suppose that each vertex of S has a token placed on it. The tokens are allowed to be moved, one at a time, by sliding along the edges of G while maintaining the property that after each move, the vertices having tokens always form an independent set of G. We would like to determine whether the tokens can be eventually brought to stay on the vertices of another independent set S' of G in this manner. In other words, we would like to decide if we can transform S into S' through a sequence of steps, each of which involves substituting a vertex in the current independent set with one of its neighbours to obtain another independent set. This problem of determining if one independent set of a graph "is reachable" from another independent set of it is known to be PSPACE-hard even for split graphs, planar graphs, and graphs of bounded treewidth. Polynomial time algorithms have been obtained for certain graph classes like trees, interval graphs, claw-free graphs, and bipartite permutation graphs. We present a polynomial time algorithm for the problem on block graphs, which are the graphs in which every maximal 2-connected subgraph is a clique. Our algorithm is the first generalization of the known polynomial time algorithm for trees to a larger class of graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Token sliding independent set reconfiguration, block graphs, polynomial time algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.31

Related Version *Full Version*: `https://arxiv.org/abs/2410.07060` [15]

Funding *Veena Prabhakaran*: DST SERB grant no: PDF/2022/002117 and the IITM Pravartak Technologies Foundation.

1 Introduction

A *reconfiguration problem* on a graph G looks at how two feasible solutions to a computational problem on G relate to one another (all graphs considered in this paper are simple and undirected unless otherwise mentioned). It involves determining, given a graph G and two feasible solutions of some computational problem on G, whether there is a step-by-step transformation from one solution to the other through a series of intermediate solutions adhering to a set of rules. Note that each intermediate solution also has to be a feasible solution to the same computational problem on G. Reconfiguration problems have been studied for various computational problems like the independent set problem [18, 34, 6, 26], dominating set problem [14, 9], vertex cover problem [28, 32, 26], matching problem [4, 26, 33], and vertex colouring problem [18, 11].

* A major part of the work that led to the results in this manuscript was done when both authors were affiliated with the Indian Statistical Institute, Chennai Centre

© Mathew C. Francis and Veena Prabhakaran;
licensed under Creative Commons License CC-BY 4.0

The "independent set reconfiguration" problem can be defined as follows. Suppose that G is a graph and C, C' are two independent sets of G. Imagine that each vertex in C has a token placed on it. We would like to determine if there is a sequence of transformation steps that can be applied on the set of tokens so that the tokens eventually are on the vertices of C'. Each transformation step must be in accordance with a predetermined set of rules, and at the end of each move, the vertices having tokens on them have to again form an independent set of G. Based on the set of rules governing the transformation steps, mainly three types of independent set reconfiguration problems have been studied in the literature – namely, the "token sliding" [5, 31], the "token jumping" [27, 31] and the "token addition/removal" [31] independent set reconfiguration problems. (Note that these three models have been studied also for reconfiguration problems other than the independent set reconfiguration problem [29, 28, 17, 16].) It has been shown that the token jumping and the token addition/removal models are equivalent for independent set reconfiguration [30]. In the token sliding model introduced by Hearn and Demaine [18], which will be the focus of our study, the only allowed transformation step is the movement of a token from the vertex it is on to one of its neighbours (the token can be imagined to be "sliding" along the edge between them).

Formally, in the token sliding model for the independent set reconfiguration problem, given a graph G and two independent sets C, C' of G, it has to be determined if there exists a sequence of independent sets $C = C_0, C_1, \ldots, C_k = C'$, where $k \geq 0$, such that for each $i \in \{0, 1, \ldots, k-1\}$, $C_{i+1} = (C_i \setminus \{u\}) \cup \{v\}$, for some $u \in C_i$ and edge uv of G. Note that if such a sequence exists then $|C_0| = |C_1| = \cdots = |C_k|$, and therefore the problem is trivial if $|C| \neq |C'|$ (in this case, C cannot be transformed into C'). So it is customary to assume that the two input independent sets are of the same cardinality. We call this decision problem TS-Ind-set reconfig, which we define formally in Section 2.

The problem TS-Ind-set reconfig was first observed to be PSPACE-complete for general graphs by Hearn and Demaine [18]. In fact, their result implies that the problem is PSPACE-complete even for subcubic planar graphs (see [7, 30]). Later, the problem was shown to be PSPACE-complete for perfect graphs by Kamiński, Medvedev and Milanič [30], and this was further improved by Lokshtanov and Mouawad [31], who showed that the problem remains PSPACE-hard even for bipartite graphs. It was shown that the problem is PSPACE-hard also for another subclass of perfect graphs called split graphs by Belmonte et al. [2]. Note that split graphs form a subclass of chordal graphs and even hole-free graphs, and hence the problem is PSPACE-complete for chordal graphs and even-hole free graphs as well. Wrochna [34] showed that the problem is PSPACE-complete when restricted to graphs of bounded bandwidth, which implies that the problem is PSPACE-complete for graphs of bounded treewidth, or in fact bounded pathwidth.

Demaine et al. [12] showed that TS-Ind-set reconfig is polynomial time solvable for trees. Polynomial time algorithms for the problem were obtained for claw-free graphs by Bonsma, Kamiński and Wrochna [8], for P_4-free graphs by Kamiński, Medvedev and Milanič [30], for interval graphs by Bonamy and Bousquet [3], and for bipartite permutation graphs and bipartite distance-hereditary graphs by Fox-Epstein et al. [14].

Our contribution

A "block" of a graph G is defined to be a maximal connected subgraph H of G such that H is a graph without any cut-vertices. (Note that it follows from this definition that if an edge e of G is not contained in any cycle of G, then e together with its endpoints forms a block of G.) A *block graph* is an undirected graph in which each block is a clique. A simple way to

visualize the structure of these graphs is through the following equivalent definition: every complete graph is a block graph, and every graph that is obtained by taking the disjoint union of a block graph G and a complete graph G' and then identifying (i.e. merging) a vertex in G with a vertex in G' is also block graph. Please refer Figure 3(a) for an example of a block graph and Section 2.2 for a formal definition of this graph class. Note that block graphs form a subclass of chordal graphs and a superclass of trees.

We prove that TS-Ind-set reconfig is polynomial-time solvable on block graphs. It is worth noting that our algorithm is the first generalization of the polynomial time algorithm of Demaine et al. for trees, since none of the other classes for which polynomial time algorithms for the problem have been obtained contains the class of trees. It was claimed in [23] and [21] that the problem can be solved in polynomial time on cactus graphs and block graphs respectively, but the authors of both papers have later announced that the algorithms are incorrect [20, 19] (also see [22, 24]), and that the complexity of the problem on both classes of graphs remains unresolved.

In this paper, we present an algorithm that takes as input a block graph G having b blocks, along with two independent sets C_1, C_2 of G, and determines in $O(b^4)$ time whether the independent set C_1 can be transformed into the independent set C_2 using token sliding. Note that b is at most $|V(G)|$ and hence this algorithm runs in time $O(|V(G)|^4)$.

Our algorithm is a generalization of the polynomial-time algorithm for trees given in [12]. The algorithm for trees first determines which tokens are "rigid" in both input independent sets. A token in an independent set is said to be rigid if it cannot be moved from its position by any sequence of valid token movements. Determining which tokens in an independent set are rigid is not very difficult for trees – a token on a vertex u is rigid if and only if every neighbour v of u has a neighbour x other than u itself that has a token that is rigid for the tree containing x in the forest $T - \{v\}$. This implies that it can be determined in linear time whether a token in a given independent set is rigid. Clearly, two independent sets whose set of vertices containing rigid tokens are unequal cannot be reconfigured with each other. Even if the vertices having rigid tokens is the same for both independent sets, if the two independent sets have an unequal number of tokens in any connected component that is obtained by the removal of the vertices containing the rigid tokens and their neighbours, it can be concluded that the independent sets are not reconfigurable with each other. It is shown in [12] that these two necessary conditions are also sufficient, i.e. any two independent sets that satisfy these two conditions can be reconfigured with each other. Thus it can be determined in linear time whether two independent sets in a tree are reconfigurable with each other. Further, their proof yields an algorithm that outputs a reconfiguration sequence of length $O(n^2)$ between the two input independent sets. Note that the result above implies that any two independent sets in a tree that have the same cardinality and contain no rigid tokens are reconfigurable with each other.

Even though similar to trees at first sight, the problem on block graphs presents many difficulties that hinder the construction of a polynomial-time algorithm similar to the one in [12]. For example, unlike for trees, two independent sets in a block graph that have the same cardinality and have no rigid tokens may not be reconfigurable with each other; see Figure 12 of [12]. This hints that instead of rigidity of tokens, one should perhaps find out which tokens are "trapped" within the blocks they are in (this is similar to the notion of "confined" tokens in [23]). However, determining which tokens are trapped in their blocks turns out to be not so straightforward.

A more important difference is perhaps the fact that in a block graph, there can be situations in which to insert one additional token into a branch of the block graph, an arbitrary number of tokens may first need to be taken out of that branch – a situation that

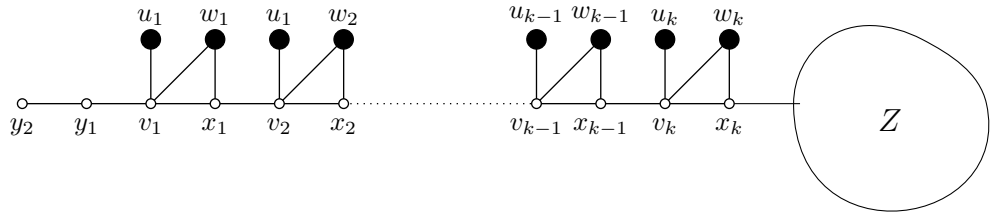


Figure 1 Diagram showing tokens placed in a part of a block graph (tokens are the solid black circles).

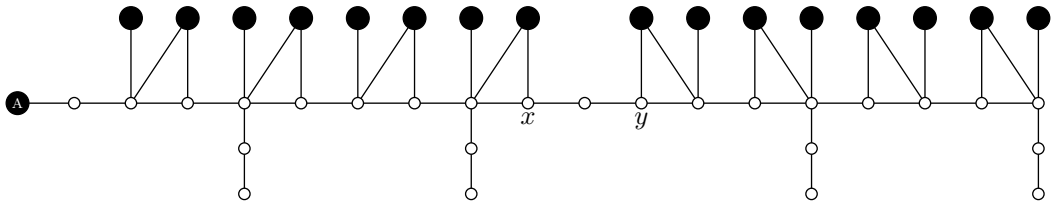


Figure 2 Token A can be moved out of its block.

never arises for a tree. An example of this phenomenon can be demonstrated as follows. In Figure 1, a block graph with some tokens placed on the vertices of an independent set is shown. In the figure, $2k$ of the tokens are shown as solid black circles. There might be an arbitrary number of tokens in the set Z of vertices. For each $i \in \{1, 2, \ldots, k\}$, let $A_i = \{y_1, y_2\} \cup \bigcup_{j \in \{1,2,\ldots,i\}} \{x_j, u_j, v_j, w_j\}$. It can be proved that in order to place a token on y_2, we will have to go through an intermediate independent set in which there are at most k tokens in A_k (which means that at least k tokens that were originally in A_k are outside A_k at this point). This can be easily proved by induction on k. If $k = 1$, then clearly, before reaching an independent set with a token on y_2, we will have to reach an independent set with a token on v_1, at which point there will be at most one token in A_1. Let us assume inductively that the statement is true for $k - 1$. Suppose that through a sequence of reconfiguration steps, we place a token on y_2. Then we know by the inductive hypothesis that there is some intermediate independent set in which there are at most $k - 1$ tokens in A_{k-1}. Consider the first such independent set in the sequence of reconfiguration steps. By the choice of this independent set, it is clear that there is a token on v_k in this independent set, which implies that there are no tokens on any other vertex of $A_k \setminus A_{k-1}$. This means that there are at most k tokens in A_k in this independent set, and we are done.

Using the construction described above, one can create situations in which it does not seem straightforward to determine whether a token is "trapped" within its block. For example, in the situation shown in Figure 2, the token A can be moved out of the block in which it is in, but determining that fact does not seem to be as easy as it is for the case of trees (the reader is encouraged to try to find a sequence of token movements that can achieve this; one such strategy is shown in Figure 6 at the end). It can be seen from a careful study of our strategy, shown in Figure 6, that we are alternately "freeing up" more and more space on the "left" and "right" sides of the graph by using the space that has already been freed up on the other side. Each time, the newly created space allows us to place a token on one of the four bottom-most vertices.

We hope that the strategy described above will serve as a motivational example for understanding Algorithm 1, which is the major part of our polynomial time algorithm for TS-Ind-set reconfig on block graphs. A sketch of the algorithm is as follows. Given a

block graph G and a configuration of tokens occupying the vertices of an independent set I of G, the algorithm determines for each branch of G how many additional tokens (relative to the number of tokens in that branch in the starting configuration I) can be moved into that branch by any sequence of token movements, assuming that any number of tokens can actually be brought to enter the branch from other parts of the graph. Once these values are obtained for each branch of G, we can use them to determine "rigid vertices" corresponding to the independent set I, which are vertices on which a token can never be placed by any sequence of token movements, starting from the configuration I (note that these are different from the "rigid tokens" in [12]). We then show that similar to the algorithm of Demaine et al. [12] for trees, two independent sets I_1 and I_2 in a block graph are reconfigurable between each other if and only if the set of rigid vertices for each of these independent sets is exactly the same, and in the graph obtained by the removal of the rigid vertices, each connected component contains an equal number of tokens in both I_1 and I_2.

Note that the above description of the algorithm is only intended to develop intuition; we now give the formal definitions of the concepts that were alluded to above, which will enable us to state our algorithm and prove its correctness rigorously.

Note. Due to space constraints, most of the intermediate lemmas required for proving the correctness of the algorithm have been stated without proofs. Please refer to the full version [15] of this paper for their proofs.

2 Notations and preliminaries

As usual, we let $V(G)$ and $E(G)$ denote the vertex set and edge set of a graph G. An edge between vertices $u, v \in V(G)$ is denoted as uv. For $S \subseteq V(G)$, we denote by $G - S$ the graph obtained by removing the vertices in S from G; i.e. $V(G - S) = V(G) \setminus S$ and $E(G - S) = E(G) \setminus \{uv \in E(G) \colon u \in S\}$. Similarly if $F \subseteq E(G)$, then we denote by $G - F$ the graph obtained by removing the edges in F from G; i.e. $V(G - F) = V(G)$ and $E(G - F) = E(G) \setminus F$. For $S \subseteq V(G)$, we let $G[S] = G - (V(G) \setminus S)$. The graph $G[S]$ is commonly known as the "subgraph that is induced in G by S". A subgraph H of G is said to be an *induced subgraph* of G if $H = G[S]$ for some $S \subseteq V(G)$. For a graph G and $u \in V(G)$, we denote by $N_G(u)$ the neighbourhood of u in G. For $S \subseteq V$, we define $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$. When H is a subgraph of G, we abbreviate $N_G(V(H))$ to just $N_G(H)$.

2.1 Reachability

Let G be a graph. We define a binary relation $\to$ on the set of independent sets of G which has the property that for two independent sets I_1 and I_2 of G, we have $I_1 \to I_2$ if and only if I_1 can be transformed into I_2 using at most one token movement. Formally, for two independent sets I_1, I_2 of G, we say that "$I_1 \to I_2$" if one of the following holds:

- $I_1 = I_2$, or
- $\exists v_1, v_2 \in V(G)$ such that $I_1 \setminus I_2 = \{v_1\}$, $I_2 \setminus I_1 = \{v_2\}$ and $v_1 v_2 \in E(G)$.

Note that the relation $\to$ is symmetric and reflexive. For independent sets I, I' of a graph G, we say that I' *is reachable from* I in G if there exist independent sets $I_0, I_1, \ldots, I_k$ of G, where $k \geq 0$, such that $I = I_0 \to I_1 \to \cdots \to I_k = I'$. Clearly, the relation "is reachable from" is an equivalence relation on the set of independent sets of G. Also, since the relation $\to$ preserves cardinality, i.e. since $|I_1| = |I_2|$ if $I_1 \to I_2$, we have that two independent sets of G that are reachable from each other are of the same cardinality.

The decision problem TS-Ind-set reconfig is defined as follows.

TS-Ind-set reconfig	
Input:	A graph G, independent sets C_1, C_2 of G
Output:	"Yes", if C_1 is reachable from C_2 in G, "No", otherwise

The following observations, which are implicitly assumed in the literature about the token sliding independent set reconfiguration problem, are easy to see (see for example Observation 4.1 in [1]).

▶ **Observation 1.** *Let G be any graph and I_1, I_2 be independent sets of G. Then I_1 is reachable from I_2 if and only if $I_1 \cap V(H)$ is reachable from $I_2 \cap V(H)$ for every connected component H of G.*

▶ **Observation 2.** *Let G be any graph, H a subgraph of G, and I_1, I_2 be independent sets of H. If I_1 is reachable from I_2 in H, then I_1 is reachable from I_2 in G.*

2.2 Block graphs

A *cut-vertex* of a graph G is a vertex $u \in V(G)$ such that $G - \{u\}$ has more connected components than G. A *block* in a graph G is a maximal subgraph of G which is isomorphic to a connected graph without any cut-vertices. For the sake of brevity of notation, we consider a block as just the set of vertices that make up the block. Thus, for us, a block of a graph G is a maximal set $S \subseteq V(G)$ such that $G[S]$ is a connected graph that does not have any cut-vertices. A *block graph* is a graph in which each block is a clique. The following folklore observation can be easily seen to follow from the definition of a block graph.

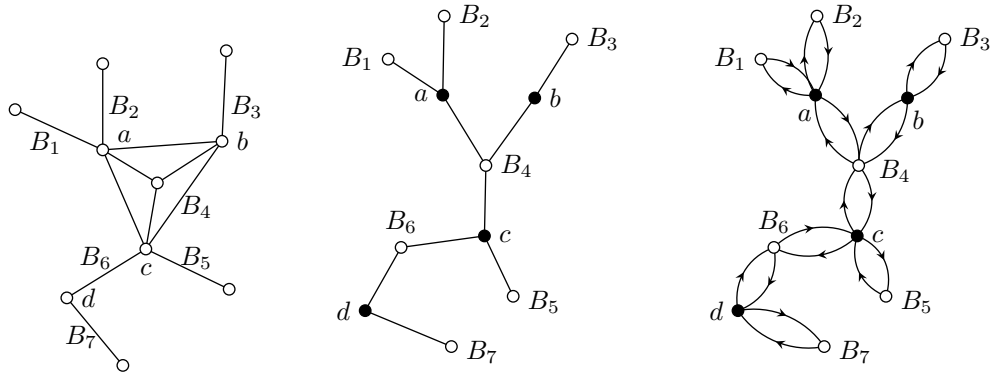
▶ **Observation 3.** *If G is a block graph, then every induced subgraph of G is also a block graph.*

By Observation 1, it is enough to solve the TS-Ind-set reconfig problem on connected graphs. Let G be a connected block graph. Note that every vertex of G is contained in some block of G. The vertices of G that are contained in more than one block are exactly the cut-vertices of G. Every pair of adjacent vertices in G belongs to a unique block of G. We denote by $V_{cut}(G)$ the set of cut-vertices of G and by $\mathcal{B}(G)$ the set of blocks of G. It is easy to see that if $u \in V(G) \setminus V_{cut}(G)$, then there is a unique block in $\mathcal{B}(G)$ that contains u. For any $u \in V(G)$, we denote by $\mathcal{B}_u(G)$ the set of all blocks in $\mathcal{B}(G)$ that contain u. For any $B \in \mathcal{B}(G)$, we define $K_B(G) = B \cap V_{cut}(G)$, i.e. $K_B(G)$ is the set of cut-vertices of G that also belong to B.

A block graph is commonly represented by its *block-tree* which is a tree having vertex set $V_{cut}(G) \cup \mathcal{B}(G)$ and edge set $\{uB \colon u \in V_{cut}(G), B \in \mathcal{B}_u(G)\}$ (see Figure 3). It is folklore that this definition indeed gives a tree (see for example Section 3.1 of [13]).

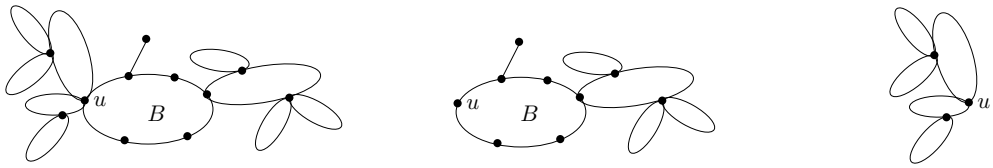
For a connected block graph G, let $\mathcal{T}(G)$ denote its block-tree with each undirected edge replaced with a pair of directed edges, one pointing in each direction. Thus $E(\mathcal{T}(G)) = \{(u, B), (B, u) \colon u \in V_{cut}(G), B \in \mathcal{B}_u(G)\}$. For ease of notation we let $\mathcal{P}_G = E(\mathcal{T}(G))$. For each $p \in \mathcal{P}_G$, we now define an induced subgraph $G[p]$ of G as follows.

For $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$, we define $G[(u, B)]$ (respectively $G[(B, u)]$) as the connected component containing u in the graph $G - \{uv \in E(G) \colon v \in B\}$ (resp. $G - \{uv \in E(G) \colon v \notin B\}$). Clearly, $G[(u, B)]$ and $G[(B, u)]$ are connected graphs. By Observation 3, we have that $G[(u, B)]$ and $G[(B, u)]$ are both block graphs. Most of the time, we abbreviate



(a) G (b) Block tree of G (c) $\mathcal{T}(G)$

Figure 3 A block graph G having four cut-vertices a, b, c, d and seven blocks $B_1, B_2, \ldots, B_7$, its corresponding tree representation and $\mathcal{T}(G)$. In (b) and (c), the black vertices represent the cut-vertices of G and the white vertices the blocks of G.



(a) G (b) $G[B, u]$ (c) $G[u, B]$

Figure 4 A schematic diagram of a block graph G, and its two subgraphs corresponding to a cut-vertex u and a block B containing u.

$G[(u, B)]$ and $G[(B, u)]$ to just $G[u, B]$ and $G[B, u]$ respectively (see Figure 4 for a schematic diagram of an example block graph G and two induced graphs $G[u, B]$ and $G[B, u]$ of it). We also abbreviate $\mathcal{B}_u(G[u, B])$ and $K_B(G[B, u])$ to $\beta_G(u, B)$ and $\kappa_G(B, u)$ respectively. We omit the subscript G when the graph being considered is clear from the context.

We shall sometimes write just $\mathcal{P}$ instead of $\mathcal{P}_G$, when the graph G is clear from the context. Let G be a connected block graph, $u \in V_{cut}(G)$, $B \in \mathcal{B}_u(G)$ and $p \in \{(u, B), (B, u)\}$. We say that u "is the base of p". Further, if $p = (u, B)$, then we let $\overline{p} = (B, u)$ and if $p = (B, u)$, then we let $\overline{p} = (u, B)$.

For $p \in \mathcal{P}_G$, a set $C \subseteq V(G[p])$ that is an independent set of G is called a *p-independent set* of G. Further, we define $\overline{C} = C \setminus \{u\}$, where u is the base of p. For an independent set S of G, we let $S[p]$ denote $V(G[p]) \cap S$. Notice that for an independent set S of G, the set $S[p]$ is a p-independent set of G. Depending on the context, we shall sometimes consider a p-independent set of G to be an independent set of G as well. Given an independent set (or p-independent set) C of G, we say that a vertex u is *under attack* in C if $N_G(u) \cap C \neq \emptyset$. Note that if $u \in C$, then u is not under attack in C.

3 Some definitions

We first define, for a connected block graph G and for each $p \in \mathcal{P}_G$, a non-negative integer $d_G(p)$ which we shall call the "depth" of p in G.

▶ **Definition 4.** *Let G be a connected block graph and $p \in \mathcal{P}_G$. If $p = (B, u)$, for some $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$, and $G[B, u]$ is a complete graph (equivalently, $\kappa_G(B, u) = \emptyset$), then we define $d_G(p) = d_G(B, u) = 0$. Otherwise, we define:*

$$d_G(p) = \begin{cases} 1 + \max\{d_G(v, B) \colon v \in \kappa_G(B, u)\} & \text{if } p = (B, u) \text{ for some } u \in V_{cut}(G) \text{ and } B \in \mathcal{B}_u(G) \\ 1 + \max\{d_G(B', u) \colon B' \in \beta_G(u, B)\} & \text{if } p = (u, B) \text{ for some } u \in V_{cut}(G) \text{ and } B \in \mathcal{B}_u(G) \end{cases}$$

Note that as before, we abbreviate $d_G(p)$ to just $d(p)$ when the graph G is clear from the context.

Similarly, for each $p \in \mathcal{P}_G$, we define a Boolean value $ua_G(p)$ inductively as follows. (Intuitively, if $ua_G(p) = True$, it roughly indicates that the base u of p is under attack in any p-independent set of G in which the tokens cannot be moved away from u so as to create space in $G[p] - \{u\}$ for more tokens to enter via u.)

▶ **Definition 5.** *Let G be a connected block graph and $p \in \mathcal{P}_G$.*
If $d_G(p) = 0$ then we define $ua_G(p) = True$.
If $p = (B, u)$ for some $u \in V_{cut}(G)$, $B \in \mathcal{B}_u(G)$, and $d_G(p) > 0$, we define:

$$ua_G(p) = ua_G(B, u) = \neg \left(\bigwedge_{\forall v \in \kappa_G(B,u)} ua_G(v, B) \wedge (B = \kappa_G(B, u) \cup \{u\}) \right)$$

On the other hand, if $p = (u, B)$ for some $u \in V_{cut}(G)$, $B \in \mathcal{B}_u(G)$, we define:

$$ua_G(p) = ua_G(u, B) = \bigvee_{\forall B' \in \beta_G(u,B)} ua_G(B', u)$$

As before, we drop the subscript G from $ua_G(p)$ when the graph under consideration is clear from the context.

Note. While evaluating an arithmetic expression that contains Boolean values, we replace all $True$ values with 1 and all $False$ values with 0.

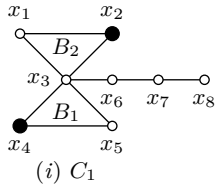
Suppose that C is a p-independent set of a connected block graph G for some $p \in \mathcal{P}_G$. We define the "capacity of C", denoted by $cap(C)$, inductively as follows. (Intuitively, the capacity of a p-independent set roughly corresponds to the number of new tokens that can be inserted via the base u of p into $G[p] - \{u\}$ while ensuring that no token in $G[p] - \{u\}$ moves towards u.)

▶ **Definition 6.** *Let G be a connected block graph, $p \in \mathcal{P}_G$ and C a p-independent set of G.*
If $p = (B, u)$ for some $u \in V_{cut}(G)$, $B \in \mathcal{B}_u(G)$, then we define:

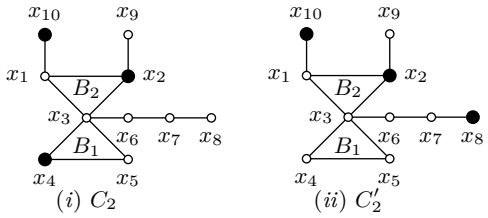
$$cap(C) = \sum_{v \in \kappa_G(p)} cap(C[v, B]) + ua_G(p) - |B \cap \overline{C}|.$$

If $p = (u, B)$ for some $u \in V_{cut}(G)$, $B \in \mathcal{B}_u(G)$, we define:

$$cap(C) = \begin{cases} 0 & \text{if } \exists B' \in \mathcal{B}_u[p] \colon cap(C[B', u]) = 0 \text{ and } ua_G(B', u) = True \\ \sum_{B' \in \beta_G[p]} \big(cap(C[B', u]) - ua_G(B', u)\big) + ua_G(p) & \text{otherwise} \end{cases}$$



(a) A block graph G_1 with two independent sets C_1 and C_1' which are not reconfigurable with each other. Here, $cap(C_1[B_1, x_3]) = cap(C_1[B_2, x_3]) = 0$ and $ua_{G_1}(B_1, x_3) = ua_{G_1}(B_2, x_3) = True$.



(b) A block graph G_2 with two independent sets C_2 and C_2' which are reconfigurable with each other. Here, $cap(C_2[B_1, x_3]) = cap(C_2[B_2, x_3]) = 0$, $ua_{G_2}(B_1, x_3) = True$ and $ua_{G_2}(B_2, x_3) = False$.

Figure 5 Effect of the parameters $cap(C[p])$ and $ua_G(p)$ on reachability.

Figure 5 depicts how the parameters $cap(C[p])$ and $ua_G(p)$, for some $p \in \mathcal{P}_G$, affect the possibility of moving a token to the base of p. Consider the graph G_1 and the independent sets C_1, C_1' in Figure 5a. In this example, the values $cap(C_1[B_2, x_3]) = 0$ and $ua_{G_1}(B_2, x_3) = True$, together impose some properties (see Lemma 1(ii) of [15]) on $C_1[B_2, x_3]$ that prevent the token on x_4 from reaching x_3, which makes C_1 and C_1' not reconfigurable with each other. Now consider the graph G_2 and the independent sets C_2, C_2' in Figure 5b. In G_2, since $ua_{G_2}(B_2, x_3) = False$, $C_2[B_2, x_3]$ exhibits a different property (see Lemma 3 of [15]) that allows the token on x_4 to reach x_3, and in turn x_8, making C_2' reachable from C_2.

▶ **Observation 7.** *Let G be a connected block graph, $p \in \mathcal{P}_G$, and C a p-independent set of G. If $d_G(p) = 0$, then $cap(C) = 1 - |\overline{C}|$.*

Proof. From Definition 4, we know that there exists $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$ such that $p = (B, u)$ and $G[p]$ is a complete graph. Note that this means that $V(G[p]) = B$. It follows directly from Definition 5 that $ua_G(p) = True$. From Definition 6 and the fact that $\kappa_G(p) = \emptyset$, we then have that $cap(C) = 1 - |B \cap \overline{C}|$. Since $V(G[p]) = B$, we have $B \cap \overline{C} = \overline{C}$. Thus $cap(C) = 1 - |\overline{C}|$. ◀

Let G be a connected block graph. Let $u \in V_{cut}(G)$ and $\mathcal{A} \subseteq \mathcal{B}_u(G)$. We say that an independent set C' of G is $(\mathcal{A}, u)$-*reachable* from an independent set C of G if there exists independent sets $C_0, C_1, \ldots, C_k$ of G, where $k \geq 0$, such that $C = C_0 \to C_1 \to \cdots \to C_k = C'$ and for each $B \in \mathcal{B}_u(G) \setminus \mathcal{A}$, $C_0[B, u] = C_1[B, u] = \cdots = C_k[B, u]$. Observe that the relation "is $(\mathcal{A}, u)$-reachable from" is also an equivalence relation defined on the independent sets of G. We shall write "(B, u)-reachable" as a shorthand for $(\{B\}, u)$-reachable and "(u, B)-reachable" as a shorthand for $(\mathcal{B}_u(G) \setminus \{B\}, u)$-reachable. Notice that if $p \in \mathcal{P}_G$, and C, C' are independent sets of G such that C' is p-reachable from C, then there exist independent sets $C_0, C_1, \ldots, C_k$, where $k \geq 0$, such that $C = C_0 \to C_1 \to \cdots \to C_k = C'$ and $C_0[\overline{p}] = C_1[\overline{p}] = \cdots = C_k[\overline{p}]$.

▶ **Lemma 8.** *Let $C_0, C_1, \ldots, C_k$, where $k \geq 0$, be independent sets of G such that $C_0 \to C_1 \to \cdots \to C_k$, and $u \in V_{cut}(G)$. If $u \notin C_0 \cup C_1 \cup \cdots \cup C_k$, then for each $B \in \mathcal{B}_u(G)$, we have $|\overline{C_0[B, u]}| = |\overline{C_k[B, u]}|$.*

4 Potentials: the global perspective

▶ **Definition 9.** *For a connected block graph G, and $p \in \mathcal{P}_G$, we define the* potential *of p with respect to an independent set C of G, denoted by $pot_G(C, p) = \max\{cap(C'[p]) + |\overline{C'[p]}| - |\overline{C[p]}| : C'$ is an independent set of G that is reachable from $C\}$.*

Algorithm 1 The algorithm for computing potentials.

```
1  Procedure COMPUTE-POTENTIALS (G, C)
       Input  : A block graph G and an independent set C of G
       Output : y[p] ∈ ℕ, ∀p ∈ 𝒫_G
2      for p ∈ 𝒫_G do
3          y[p] ← 0
4      end
5      updated ← True
6      while updated do
7          updated ← False
8          for p ∈ 𝒫_G do
9              y' ← y[p]
10             if p = (B, u), for some u ∈ V_cut(G) and B ∈ ℬ_u(G) then
11                 y' ← Σ_{∀u' ∈ κ_G(B,u)} y[u', B] + ua_G(B, u) − |B ∩ C[B, u]|
12             end
13             else if p = (u, B), for some u ∈ V_cut(G), B ∈ ℬ_u(G), and
                    ∄B', B'' ∈ ℬ_u(G) such that y[B', u] = y[B'', u] = 0 and
                    ua_G(B', u) = ua_G(B'', u) = True then
14                 y' ← Σ_{∀B' ∈ β_G(u,B)} ( y[B', u] − ua_G(B', u) ) + ua_G(u, B)
15             end
16             if y[p] < y' then
17                 y[p] ← y'
18                 updated ← True
19                 break
20             end
21         end
22     end
23     return y
```

When the graph G is clear from the context, we sometimes abbreviate $pot_G(C, p)$ to just $pot(C, p)$. We claim that the procedure COMPUTE-POTENTIALS, listed as Algorithm 1, when given a connected block graph G and an independent set C of it, computes the value of $pot_G(C, p)$ for each $p \in \mathcal{P}_G$.

For the rest of this section, we assume that G is a connected block graph and C is an independent set of G. For $p \in \mathcal{P}$, let $\mathbf{x}[p]$ denote the final value of the variable $y[p]$ that is computed by the procedure COMPUTE-POTENTIALS(G, C). Our aim in this section will be to prove that for each $p \in \mathcal{P}$, $\mathbf{x}[p] = pot(C, p)$.

Let us analyze Algorithm 1 in detail. Suppose that the **while** loop starting on line 6 gets executed t times in total during the execution of Algorithm 1. For each $i \in \{1, 2, \ldots, t\}$ and $p \in \mathcal{P}$, we let $y^{(i)}[p]$ denote the value of the variable $y[p]$ just before the i-th iteration of the **while** loop starts. Since it is clear that no variable $y[p]$, for $p \in \mathcal{P}$, is updated during the last iteration of the **while** loop, it follows from the definition of $\mathbf{x}[p]$ that for each $p \in \mathcal{P}$, $y^{(t)}[p] = \mathbf{x}[p]$.

It is easy to see that the algorithm never decreases the value of a variable $y[p]$, for any $p \in \mathcal{P}$, and that exactly one of the variables $y[p]$, for $p \in \mathcal{P}$, gets updated during every iteration of the **while** loop other than the last iteration. Thus we have the following two observations.

▶ **Observation 10.** *Let $p \in \mathcal{P}$. For $1 \leq i < j \leq t$, $y^{(i)}[p] \leq y^{(j)}[p]$, and therefore,* $\mathbf{x}[p] = y^{(t)}[p] \geq y^{(1)}[p] = 0$.

▶ **Observation 11.** *For every $i \in \{2, 3, \ldots, t\}$, there exists $p' \in \mathcal{P}$ such that $y^{(i)}[p'] > y^{(i-1)}[p']$ and $y^{(i)}[p] = y^{(i-1)}[p]$ for every $p \in \mathcal{P} \setminus \{p'\}$.*

Consider the last iteration of the **while** loop. Since this is the last iteration of the **while** loop, line 18 is not executed during this iteration. This means that the **break** statement of line 19 is never executed during this iteration, which further implies that the **for** loop of line 8 gets executed for every $p \in \mathcal{P}$.

▶ **Lemma 12.** *For any $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$,*

$$\mathbf{x}[B, u] = \sum_{\forall u' \in \kappa(B,u)} \mathbf{x}[u', B] + ua(B, u) - |B \cap \overline{C[B, u]}|$$

▶ **Lemma 13.** *For any $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$,*
if $\exists B', B'' \in \mathcal{B}_u(G)$ such that $\mathbf{x}[B', u] = \mathbf{x}[B'', u] = 0$ and $ua(B', u) = ua(B'', u) = True$, then

$$\mathbf{x}[u, B] = 0$$

and otherwise,

$$\mathbf{x}[u, B] = \sum_{\forall B' \in \beta(u,B)} \Big(\mathbf{x}[B', u] - ua(B', u)\Big) + ua(u, B)$$

4.1 Time complexity of the algorithm

▶ **Lemma 14.** *For any $p \in \mathcal{P}$, $\mathbf{x}[p] \leq |\mathcal{B}(G[p])|$.*

Proof. We prove this by induction on $d(p)$. For the base case, observe that if $d(p) = 0$, then $p = (B, u)$ for some $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$, $ua(B, u) = True$, and $G[B, u]$ is a complete graph. Then we have by Lemma 12 that $\mathbf{x}[p] = 1 - |B \cap \overline{C[B, u]}| \leq 1 = |\mathcal{B}(G[B, u])|$. This proves the base case. For the inductive step, let us assume that for all $p' \in \mathcal{P}$ such that $d(p') < d(p)$, the statement of the lemma is true. Suppose that $p = (B, u)$ for some $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$. Then from Lemma 12, we have that $\mathbf{x}[B, u] = \sum_{\forall u' \in \kappa(B,u)} \mathbf{x}[u', B] + ua(B, u) - |B \cap \overline{C[B, u]}|$. Since $d(u', B) < d(B, u)$ for all $u' \in \kappa(B, u)$, we can use the induction hypothesis to conclude that $\mathbf{x}[B, u] \leq \sum_{\forall u' \in \kappa(B,u)} |\mathcal{B}(G[u', B])| + ua(B, u) - |B \cap \overline{C[B, u]}|$. It is easy to see that $|\mathcal{B}(G[B, u])| = 1 + \sum_{\forall u' \in \kappa(B,u)} |\mathcal{B}(G[u', B])|$. We thus have that $\mathbf{x}[B, u] \leq |\mathcal{B}(G[B, u])| - 1 + ua(B, u) - |B \cap \overline{C[B, u]}| \leq |\mathcal{B}(G[B, u])|$. Next, suppose that $p = (u, B)$ for some $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$. Then we have by Lemma 13 that either $\mathbf{x}[u, B] = 0$ or $\mathbf{x}[u, B] = \sum_{\forall B' \in \beta(u,B)} \Big(\mathbf{x}[B', u] - ua(B', u)\Big) + ua(u, B)$. In the former case, we are done since $|\mathcal{B}(G[u, B])| \geq 0$. So we assume that $\mathbf{x}[u, B] = \sum_{\forall B' \in \beta(u,B)} \Big(\mathbf{x}[B', u] - ua(B', u)\Big) + ua(u, B)$. Since $d(B', u) < d(u, B)$ for all $B' \in \beta(u, B)$, we have by the induction hypothesis that $\mathbf{x}[u, B] \leq \sum_{\forall B' \in \beta(u,B)} \Big(|\mathcal{B}(G[B', u])| - ua(B', u)\Big) + ua(u, B)$. It can be seen from Definition 5 that if $ua(B', u) = True$ for any $B' \in \beta(u, B)$, then $ua(u, B) = True$. Since $\sum_{\forall B' \in \beta(u,B)} |\mathcal{B}(G[B', u])| = |\mathcal{B}(G[u, B])|$, it follows that $\sum_{\forall B' \in \beta(u,B)} \Big(|\mathcal{B}(G[B', u])| - ua(B', u)\Big) + ua(u, B) \leq |\mathcal{B}(G[u, B])|$. Thus, we get $\mathbf{x}[u, B] \leq |\mathcal{B}(G[u, B])|$, and we are done. ◀

▶ **Lemma 15.** *The procedure* COMPUTE-POTENTIALS(G, C) *runs in time* $O(|\mathcal{B}(G)|^4)$.

Proof. Let $n = |V_{cut}(G)|$ and $m = |\mathcal{B}(G)|$. Note that an adjacency list representation of the block tree of $\mathcal{T}(G)$ can be computed in $O(|V(G)| + |E(G)|)$ time [25]. Recall that $\mathcal{P} = E(\mathcal{T}(G))$. Thus the operations like determining $\beta(u, B)$ and $\kappa(B, u)$, for some $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$, become just adjacency checking operations on $\mathcal{T}(G)$. From Observations 10 and 11, it follows that the **while** loop of Algorithm 1 gets executed at most $\sum_{p\in\mathcal{P}} \mathbf{x}[p]$ times, which by Lemma 14 is at most $|\mathcal{P}|m$ times. It is not difficult to see that $|\mathcal{P}| = 2|E(\mathcal{T}(G))| = 2(n + m - 1)$. Thus, we have that the **while** loop gets executed at most $2m(n + m - 1)$ times. As the **for** loop gets executed at most $|\mathcal{P}|$ times inside each iteration of the **while** loop, and it takes $O(n + m)$ time for one iteration of the **for** loop, we can conclude that each iteration of the **while** loop takes time at most $O(|\mathcal{P}|(n + m)) = O((n + m)^2)$. Thus the total running time of the algorithm is $O(m(n + m)^3) = O(m^4)$ (since $|\mathcal{B}(G)| = m \geq n = |V_{cut}(G)|$). ◀

4.2 Proof of correctness

▶ **Lemma 16.** *Let $p \in \mathcal{P}$. There exists an independent set C' of G that is reachable from C such that $cap(C'[p]) \geq \mathbf{x}[p]$ and $|\overline{C'[p]}| = |\overline{C[p]}|$.*

▶ **Lemma 17.** *For every $p \in \mathcal{P}$ and every independent set C' of G that is reachable from C, we have $\mathbf{x}[p] \geq cap(C'[p]) + |\overline{C'[p]}| - |\overline{C[p]}|$.*

▶ **Corollary 18.** *For each $p \in \mathcal{P}$, $\mathbf{x}[p] = pot(C, p)$.*

Proof. Clearly, for each independent set C' of G that is reachable from C, we have from Lemma 17 that $cap(C'[p]) + |\overline{C'[p]}| - |\overline{C[p]}| \leq \mathbf{x}[p]$. It follows that $pot(C, p) \leq \mathbf{x}[p]$. From Lemma 16, we have that there exists an independent set C' of G that is reachable from C such that $cap(C'[p]) \geq \mathbf{x}[p]$ and $|\overline{C'[p]}| = |\overline{C[p]}|$. Then we have that $cap(C'[p]) + |\overline{C'[p]}| - |\overline{C[p]}| = cap(C'[p]) \geq \mathbf{x}[p]$. This implies that $pot(C, p) \geq \mathbf{x}[p]$. This completes the proof. ◀

By the above corollary, we can reword Lemmas 12, 13, and 16 as follows.

▶ **Corollary 19.** *Let G be any connected block graph and C be an independent set of G.*

(i) *For any $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$,*

$$pot_G(C, (B, u)) = \sum_{\forall u' \in \kappa_G(B,u)} pot_G(C, (u', B)) + ua_G(B, u) - |B \cap \overline{C[B, u]}|$$

(ii) *For any $u \in V_{cut}(G)$ and $B \in \mathcal{B}_u(G)$,*
if $\exists B', B'' \in \mathcal{B}_u(G)$ such that $pot_G(C, (B', u)) = pot_G(C, (B'', u)) = 0$ and $ua_G(B', u) = ua_G(B'', u) = True$, then

$$pot_G(C, (u, B)) = 0$$

and otherwise,

$$pot_G(C, (u, B)) = \sum_{\forall B' \in \beta_G(u,B)} \Big(pot_G(C, (B', u)) - ua_G(B', u)\Big) + ua_G(u, B)$$

(iii) *For each $p \in \mathcal{P}_G$, there exists an independent set C' of G that is reachable from C such that $cap(C'[p]) \geq pot_G(C, p)$ and $|\overline{C'[p]}| = |\overline{C[p]}|$.*

Also, Lemma 15 can now be reworded as:

▶ **Theorem 20.** *Given a connected block graph G and an independent set C of it, the procedure* COMPUTE-POTENTIALS(G, C) *runs in time $O(|\mathcal{B}(G)|^4)$ and computes the value $pot_G(C, p)$ for each $p \in \mathcal{P}_G$.*

5 Rigid vertices

Let G be a connected block graph and C be any independent set of G.

▶ **Definition 21.** *A vertex $u \in V_{cut}(G)$ is said to be* rigid for C*, if there exist $B, B' \in \mathcal{B}_u(G)$ such that $pot_G(C,(B,u)) = pot_G(C,(B',u)) = 0$ and $ua_G(B,u) = ua_G(B',u) = True$.*

▶ **Lemma 22.** *If $u \in V_{cut}(G)$ is rigid for an independent set C of G, then there does not exist any independent set C' of G that is reachable from C for which $u \in C'$.*

▶ **Lemma 23.** *Let $u \in V_{cut}(G)$ such that u is not rigid for C and let $B \in \mathcal{B}_u(G)$. If u is under attack in $C[u,B]$ (i.e. $|N_G(u) \cap C[u,B]| \geq 1$), then there is an independent set C' of G that is (u,B)-reachable from C such that $|N_G(u) \cap C'[u,B]| = 1$.*

▶ **Lemma 24.** *Let C_1 and C_2 be two independent sets of G and let W_1 and W_2 be the set of cut-vertices of G that are rigid for C_1 and C_2 respectively. If C_2 is reachable from C_1, then $W_1 = W_2$.*

Proof. Suppose for the sake of contradiction that C_2 is reachable from C_1 and $W_1 \neq W_2$. We can assume without loss of generality that there exists $u \in W_1 \setminus W_2$. Then we know that there exists $B \in \mathcal{B}_u(G)$ such that $ua_G(B,u) = True$, $pot_G(C_1,(B,u)) = 0$, and $pot_G(C_2,(B,u)) \geq 1$. By Corollary 19(iii), we have that there is an independent set C_2' that is reachable from C_2 such that $cap(C_2'[B,u]) \geq pot_G(C_2,(B,u)) \geq 1$. Since C_2 is reachable from C_1, we have that C_2' is also reachable from C_1. Then by Definition 9, we get that $cap(C_2'[B,u]) - |\overline{C_2'[B,u]}| + |\overline{C_1[B,u]}| \leq pot_G(C_1,(B,u)) = 0$, which implies that $|\overline{C_2'[B,u]}| - |\overline{C_1[B,u]}| \geq 1$. Let $D_0, D_1, \ldots, D_k$ be independent sets of G such that $C_1 = D_0 \to D_1 \to \cdots \to D_k = C_2'$. As $|\overline{C_2'[B,u]}| \neq |\overline{C_1[B,u]}|$, we have from Lemma 8 that there exists some $i \in \{0,1,\ldots,k\}$ such that $u \in D_i$. Then D_i is an independent set of G that is reachable from C_1 and contains the vertex u that is rigid for C_1. This contradicts Lemma 22. ◀

6 Restricting to subgraphs

In this section, we show that given a connected block graph G and an independent set C of it, there are certain kinds of induced subgraphs H of G having the property that each potential of H has the same value as it had in G.

For this section, we assume that G is a connected block graph and C is an independent set of G. Let $a \in V_{cut}(G)$ and $A \in \mathcal{B}_a(G)$. Let $H = G - V(G[a,A])$. It is easy to see that H is a connected block graph. Observe that $\mathcal{B}(H) = (\mathcal{B}(G[A,a]) \setminus \{A\}) \cup (A \setminus \{a\})$ if $|A| > 2$ and $\mathcal{B}(H) = \mathcal{B}(G[A,a]) \setminus \{A\}$ if $|A| = 2$. Observe that $V_{cut}(H) \subseteq V_{cut}(G)$. We now define a function $f_{H,G} : \mathcal{B}(H) \to \mathcal{B}(G)$. For each $B \in \mathcal{B}(H)$, we define $f_{H,G}(B) \in \mathcal{B}(G)$ as follows. If $|A| = 2$, then we simply define $f_{H,G}(B) = B$ for all $B \in \mathcal{B}(H)$. Otherwise, there exists a unique block $A' \in \mathcal{B}(H)$ such that $A' \subseteq A$. Then we define $f_{H,G}(B)$ as follows: we define $f_{H,G}(A') = A$, and for every $B \in \mathcal{B}(H) \setminus \{A'\}$, we define $f_{H,G}(B) = B$. It is easy to verify that for all $B \in \mathcal{B}(H)$, $f_{H,G}(B)$ is well defined and $f_{H,G}(B) \in \mathcal{B}(G)$. For $(u,B) \in \mathcal{P}_H$, we abuse notation and define that $f_{H,G}(u,B) = (u, f_{H,G}(B))$, and for $(B,u) \in \mathcal{P}_H$, we similarly define $f_{H,G}(B,u) = (f_{H,G}(B), u)$.

▶ **Lemma 25.** *If $a \notin C$, $pot_G(C,(a,A)) = 0$ and $ua_G(a,A) = True$, then for each $p \in \mathcal{P}_H$, we have $ua_H(p) = ua_G(f_{H,G}(p))$ and $pot_H(C \cap V(H), p) = pot_G(C, f_{H,G}(p))$.*

7 Putting the pieces together

▶ **Lemma 26.** *Let G be a connected block graph and C be an independent set of G. Let H be a connected component of the graph obtained by removing all vertices of G that are rigid for C. Then there are no vertices in H that are rigid for $C \cap V(H)$.*

▶ **Lemma 27.** *Let G be a connected block graph and C_1, C_2 be independent sets of G. If no cut-vertex of G is rigid for C_1 or C_2, and $|C_1| = |C_2|$, then C_2 is reachable from C_1.*

Proof. We prove this lemma using induction on $|C_1| = |C_2| = r$. As the base case, we assume that $r = 1$. Let $C_1 = \{u\}$ and $C_2 = \{v\}$. Suppose that $P = w_1w_2 \dots w_k$ is a path from u to v in G such that $w_1 = u$ and $w_k = v$. Since G is connected, P is guaranteed to exist. Let $D_1 = C_1$. For $i \in \{2, 3, \dots, k\}$, we define $D_i = (D_{i-1} \setminus \{w_{i-1}\}) \cup \{w_i\}$. Clearly, $C_1 = D_1 \to D_2 \to \dots \to D_k = C_2$, and hence C_2 is reachable from C_1. This proves the base case.

For the inductive step, we assume that if no cut-vertex of G is rigid for independent sets C_1' and C_2' of G, and $|C_1'| = |C_2'| < r$, then C_2' is reachable from C_1'. Now, let us prove the lemma for C_1 and C_2. We claim that there exists $l \in V_{cut}(G)$ such that there exists at most one $B \in \mathcal{B}_l(G)$ having $\kappa_G(B, l) \neq \emptyset$. Indeed, we can choose as l an endpoint of a longest path in G whose both endpoints are cut-vertices of G. Choose as L a block from $\mathcal{B}_l(G) \setminus \{B\}$. Note that $G[L, l]$ is a complete graph. Let $h \in L \setminus \{l\}$. First, we define independent sets S_1, S_2 of G such that $h \in S_1 \cap S_2$, S_1 is reachable from C_1, and S_2 is reachable from C_2.

Let $i \in \{1, 2\}$.

If $h \in C_i$, then we simply define $S_i = C_i$ and we are done. So we assume that $h \notin C_i$. Since $N_G(h) \subseteq L$, we know that $|N_G(h) \cap C_i| \leq 1$. Thus, if there exists $v \in N_G(h) \cap C_i$, then $S_i = (C_i \setminus \{v\}) \cup \{h\}$ is an independent set of G such that $C_i \to S_i$, and we are done. So we assume that $(\{h\} \cup N_G(h)) \cap C_i = \emptyset$, or in other words, every vertex in C_i is at a distance of at least 2 from h in G. Choose a vertex $v \in C_i$ for which the distance between v and h in G is as small as possible. Let $P = w_1w_2 \dots w_k$ be the shortest path from v to h in G such that $w_1 = v$ and $w_k = h$ (thus the distance between v and h in G is $k - 1$). Note that by our assumption, we have $k \geq 3$. Since G is a block graph, it can be easily seen that $\{w_2, w_3, \dots, w_{k-1}\} \subseteq V_{cut}(G)$, and also that $w_{k-1} = l$. If for some $j \in \{3, 4, \dots, k\}$, there exists a vertex $x \in N_G(w_j) \cap C_i$, then $xw_jw_{j+1} \dots w_k$ is a path in G between x and h having length $k - j + 1$. Then x is a vertex in C_i such that the distance between it and h in G is smaller than the distance between v and h in G, which contradicts the choice of v. Therefore, w_j is not under attack in C_i, or in other words $N_G(w_j) \cap C_i = \emptyset$, for any $j \in \{3, 4, \dots, k\}$.

Let $X \in \mathcal{B}(G)$ such that X contains the edge w_2w_3. Notice that w_2 is under attack in $C_i[w_2, X]$ (as $w_1 \in C_i$). By Lemma 23, there exists an independent set D that is (w_2, X)-reachable from C_i such that $|N_G(w_2) \cap D[w_2, X]| = 1$. Let $N_G(w_2) \cap D[w_2, X] = \{y\}$. Observe that $yw_2 \dots w_k$ is a path from y to $h = w_k$ in G. As $D[X, w_2] = C_i[X, w_2]$, we have that $N_G(w_j) \cap D = \emptyset$ for each $j \in \{3, 4, \dots, k\}$. Let $D_1 = D$ and $D_2 = (D_1 \setminus \{y\}) \cup \{w_2\}$. For $j \in \{3, 4, \dots, k\}$, we define $D_j = (D_{j-1} \setminus \{w_{j-1}\}) \cup \{w_j\}$. It follows from the observations above that each of $D_1, D_2, \dots, D_k$ is an independent set of G, and we also clearly have $D = D_1 \to D_2 \to \dots \to D_k$. We define $S_i = D_k$. Then S_i is reachable from D, and hence also from C_i. We also have $h = w_k \in D_k = S_i$.

Recall that $l = w_{k-1}$. Thus $l \in D_{k-1}$, and since $D_{k-1} \to D_k = S_i$, we have that $N_G(l) \cap S_i = \{h\}$. This implies that $S_i[l, B] = \{h\}$ (recall that $B \in \mathcal{B}_l(G)$ such that $\kappa_G(B, l) \neq \emptyset$). Let $H = G - V(G[l, B])$. Clearly, H is a connected block graph. Note that we can define a function $f_{H,G} : \mathcal{P}_H \to \mathcal{P}_G$ as described in the beginning of Section 6. Since each $B' \in \beta_G(l, B)$ is a complete graph, we have by Observation 7 that $ua_G(B', l) = True$,

and so in particular $ua_G(L, l) = True$. Since $h \in S_i$, this implies by Corollary 19(i) that $pot_G(S_i, (L, l)) = 0$. Moreover, since $S_i[l, B] = \{h\}$, we have that for each $B' \in \beta_G(l, B) \setminus \{L\}$, $|\overline{S_i}[B', l] \cap B'| = 0$, which implies by Corollary 19(i) that $pot_G(S_i, (B', l)) = 1$. Further, we get by Definition 5 that $ua_G(l, B) = True$, which implies by Corollary 19(ii) that $pot_G(S_i, (l, B)) = 0$. Therefore, since $l \notin S_i$, we get by Lemma 25 that for each $p \in \mathcal{P}_H$, $ua_H(p) = ua_G(f_{H,G}(p))$ and $pot_H(S_i \cap V(H), p) = pot_G(S_i, f_{H,G}(p))$.

Since S_i is reachable from C_i and no vertex is rigid for C_i, by Lemma 24 it is clear that there does not exist a vertex that is rigid for S_i. This implies that for each $u \in V_{cut}(G)$, there does not exist $B', B'' \in \mathcal{B}_u(G)$ such that $pot_G(S_i, (B', u)) = pot_G(S_i, (B'', u)) = 0$ and $ua_G(B', u) = ua_G(B'', u) = True$. From our previous observation that for each $p \in \mathcal{P}_H$, $ua_H(p) = ua_G(f_{H,G}(p))$ and $pot_H(S_i \cap V(H), p) = pot_G(S_i, f_{H,G}(p))$, we get that for each $u \in V_{cut}(H)$, there does not exist $B', B'' \in \mathcal{B}_u(H)$ such that $pot_H(S_i \cap V(H), (B', u)) = pot_H(S_i \cap V(H), (B'', u)) = 0$ and $ua_H(B', u) = ua_H(B'', u) = True$. This implies that in the graph H, no cut-vertex is rigid for the independent set $S_i \cap V(H)$ of H.

We thus have that no cut-vertex of H is rigid for either of the independent sets $S_1 \cap V(H)$ or $S_2 \cap V(H)$ of H. Further, since $|S_1| = |C_1| = |C_2| = |S_2| = r$ and $S_1[l, B] = S_2[l, B] = \{h\}$, we have $|S_1 \cap V(H)| = |S_2 \cap V(H)| = r - 1$, which implies by the induction hypothesis that in H, $S_2 \cap V(H)$ is reachable from $S_1 \cap V(H)$. This implies that there exist independent sets $R_0, R_1, \ldots, R_q$ of H, where $q \geq 0$, such that $S_1 \cap V(H) = R_0 \to R_1 \to \cdots \to R_q = S_2 \cap V(H)$. For each $j \in \{0, 1, \ldots, q\}$, since $N_G(h) \cap V(H) = \emptyset$, it is clear that $R'_j = R_j \cup \{h\}$ is an independent set of G. It is easy to see that for each $j \in \{0, 1, \ldots, q-1\}$, since $R_j \to R_{j+1}$, we also have $R'_j \to R'_{j+1}$. Moreover, as $S_1[l, B] = S_2[l, B] = \{h\}$, we have $R'_1 = S_1$ and $R'_q = S_2$. Hence, we get $S_1 = R'_0 \to R'_1 \to \cdots \to R'_q = S_2$, which shows that S_2 is reachable from S_1. Since we have already showed that S_1 is reachable from C_1 and S_2 is reachable from C_2, we get that C_2 is reachable from C_1. Hence, we are done. ◀

▶ **Theorem 28.** *Let G be a block graph and let C_1, C_2 be independent sets of G. Let W_1, W_2 be the set of vertices of G that are rigid for C_1, C_2 respectively. Then C_1 is reachable from C_2 if and only if $W_1 = W_2$, and for every connected component H of $G - W_1$, we have $|C_1 \cap V(H)| = |C_2 \cap V(H)|$.*

Proof. First, we prove the forward direction for which we consider the contrapositive statement. Suppose either $W_1 \neq W_2$ or there exists a connected component H of $G - W_1$ such that $|C_1 \cap V(H)| \neq |C_2 \cap V(H)|$. If $W_1 \neq W_2$, then by Lemma 24 we have that C_1 is not reachable from C_2 and we are done in this case. Hence, we assume that $W_1 = W_2$ and that there exists a connected component H of $G - W_1$ such that $|C_1 \cap V(H)| \neq |C_2 \cap V(H)|$. Suppose for the sake of contradiction that C_1 is reachable from C_2. Then, there exists independent sets $D_0, D_1 \ldots, D_k$, where $k \geq 0$, such that $C_1 = D_0 \to D_1 \to \cdots \to D_k = C_2$. Let $j = \min\{i \in \{0, 1, \ldots, k\} \colon |D_i \cap V(H)| \neq |C_1 \cap V(H)|\}$. Then clearly, we have $j > 0$ and $|D_{j-1} \cap V(H)| = |C_1 \cap V(H)|$. Since $D_{j-1} \to D_j$, there exists $uv \in E(G)$ such that $(D_{j-1} \setminus D_j) \cup (D_j \setminus D_{j-1}) = \{u, v\}$. As $|D_j \cap V(H)| \neq |D_{j-1} \cap V(H)|$, it must be the case that one of u, v is in $V(H)$ and the other is not in $V(H)$. Without loss of generality, assume that $u \in V(H)$ and $v \notin V(H)$. Since $uv \in E(G)$, it follows that $v \in N_G(H)$. Since H is a connected component of $G - W_1$, we now have that $v \in W_1$, or in other words, v is rigid for C_1. This means that one of D_{j-1}, D_j is an independent set of G that is reachable from C_1 that contains a vertex that is rigid for C_1. This contradicts Lemma 22.

Next, we prove the reverse direction. Suppose that $W_1 = W_2$ and for every connected component H of $G - W_1$, we have $|C_1 \cap V(H)| = |C_2 \cap V(H)|$. By Lemma 26, we have that each connected component H of $G - W_1$ has no vertex that is rigid for $C_1 \cap V(H)$ or

$C_2 \cap V(H)$. For each connected component H of $G - W_1$, since $|C_1 \cap V(H)| = |C_2 \cap V(H)|$, applying Lemma 27 to H and the independent sets $C_1 \cap V(H)$ and $C_2 \cap V(H)$ of H, we can conclude that $C_1 \cap V(H)$ is reachable from $C_2 \cap V(H)$ in H. Moreover, by Lemma 22, we know that $W_1 \cap C_1 = W_1 \cap C_2 = \emptyset$, which implies that $C_1, C_2 \subseteq V(G - W_1)$. It now follows from Observation 1 that C_1 is reachable from C_2 in $G - W_1$, and then from Observation 2 that C_1 is reachable from C_2 in G. ◀

▶ **Corollary 29.** TS-Ind-set reconfig *is polynomial time solvable on block graphs.*

Proof. Consider an input instance (G, C_1, C_2) of the TS-Ind-set reconfig problem, where G is a block graph and C_1, C_2 are independent sets of G. We can assume that the input block graph G is connected, as otherwise, we can simply solve the problem on each connected component of G (Observation 1). Note that this means that $|V(G)| \leq |E(G)|$. We first run Compute-potentials(G, C_1) and Compute-potentials(G, C_2) to compute $pot_G(C_1, p)$ and $pot_G(C_2, p)$ for all $p \in \mathcal{P}_G$. By Theorem 20, this can be done in $O(|\mathcal{B}(G)|^4)$ time. It is easy to see that once the potentials with respect to C_1 and C_2 have been computed, the set of vertices that are rigid for C_1 and the set of vertices that are rigid for C_2 can both be determined, and the equality of these sets checked, in $O(|V(G)| + |E(G)|) = O(|E(G)|)$ time. If the set of vertices that are rigid is not the same for both C_1 and C_2, we can return "No" since we know by Theorem 28 that C_2 is not reachable from C_1. Otherwise, we remove the (common) set of rigid vertices for both C_1 and C_2 to obtain a graph G' – this can be done in $O(|V(G)| + |E(G)|) = O(|E(G)|)$ time. Clearly, in $O(|E(G)|)$ time, we can determine the connected components of G' and also determine if $|C_1 \cap V(H)| = |C_2 \cap V(H)|$ for each connected component H of G'. We return "No" if there exists some connected component H of G' such that $|C_1 \cap V(H)| \neq |C_2 \cap V(H)|$, and otherwise we return "Yes". From Theorem 28, it follows that our algorithm returns "Yes" if and only if the independent set C_2 of G is reachable from C_1. Clearly, this algorithm runs in time $O(|\mathcal{B}(G)|^4)$. This completes the proof. ◀

8 Conclusion

Although Corollary 29 tells us that there is a polynomial time algorithm that can determine whether two independent sets in a block graph are reconfigurable with each other, that algorithm does not produce a reconfiguration sequence between the two independent sets. In fact, it is not even clear whether there is a reconfiguration sequence of polynomial length between any two independent sets in a block graph that are reconfigurable with each other. It was shown in Briański et al. [10] that for any two independent sets that are reconfigurable with each other in an interval graph, there is a reconfiguration sequence of length $O(n^3)$ between them (where n is the number of vertices in the graph). For trees, it is shown in Demaine et al. [12] that any two independent sets that are reconfigurable with each other have a reconfiguration sequence of length $O(n^2)$ between them.

References

1 Rémy Belmonte, Tesshu Hanaka, Michael Lampis, Hirotaka Ono, and Yota Otachi. Independent set reconfiguration parameterized by modular-width. *Algorithmica*, 82(9):2586–2605, 2020. doi:10.1007/S00453-020-00700-Y.

2 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory of Computing Systems*, 65:662–686, 2021. doi:10.1007/S00224-020-09967-8.

3 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science (WG 2017)*, volume 10520 of *Lecture Notes in Computer Science*, pages 127–139, Eindhoven, Netherlands, 2017. Springer International Publishing. doi:10.1007/978-3-319-68705-6_10.

4 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30*, volume 138 of *LIPIcs*, pages 80:1–80:14, Aachen, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.MFCS.2019.80.

5 Marthe Bonamy, Paul Dorbec, and Paul Ouvrard. Dominating sets reconfiguration under token sliding. *Discrete Applied Mathematics*, 301:6–18, 2021. doi:10.1016/j.dam.2021.05.014.

6 Paul S. Bonsma. Independent set reconfiguration in cographs and their generalizations. *Journal of Graph Theory*, 83(2):164–195, 2016. doi:10.1002/JGT.21992.

7 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.

8 Paul S. Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97, Copenhagen, Denmark, 2014. Springer. doi:10.1007/978-3-319-08404-6_8.

9 Nicolas Bousquet and Alice Joffard. TS-reconfiguration of dominating sets in circle and circular-arc graphs. In Evripidis Bampis and Aris Pagourtzis, editors, *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, September 12-15, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pages 114–134, Athens, Greece, 2021. Springer. doi:10.1007/978-3-030-86593-1_8.

10 Marcin Briański, Stefan Felsner, Jędrzej Hodor, and Piotr Micek. Reconfiguring independent sets on interval graphs. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27*, volume 202 of *LIPIcs*, pages 23:1–23:14, Tallinn, Estonia, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.23.

11 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5-6):913–919, 2008. doi:10.1016/j.disc.2007.07.028.

12 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.

13 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, Heidelberg, 2012.

14 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In Khaled M. Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation - 26th International Symposium, ISAAC 2015, December 9-11*, volume 9472 of *Lecture Notes in Computer Science*, pages 237–247, Nagoya, Japan, 2015. Springer. doi:10.1007/978-3-662-48971-0_21.

15 Mathew C. Francis and Veena Prabhakaran. Token sliding independent set reconfiguration on block graphs, 2024. doi:10.48550/arXiv.2410.07060.

16 Ruth Haas and Karen Seyffarth. The k-dominating graph. *Graphs and Combinatorics*, 30(3):609–617, 2014. doi:10.1007/S00373-013-1302-3.

17 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016. doi:10.1016/j.tcs.2016.08.016.

18 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

19 Duc A. Hoang. A note regarding "Sliding Tokens on Block Graphs". https://hoanganhduc.github.io/events/WALCOM2017/note.pdf, September 2019.

20 Duc A. Hoang. A note regarding "Sliding Tokens on a Cactus". https://hoanganhduc.github.io/events/ISAAC2016/note.pdf, July 2024.

21 Duc A. Hoang, Eli Fox-Epstein, and Ryuhei Uehara. Sliding tokens on block graphs. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *Algorithms and Computation: 11th International Conference and Workshops (WALCOM 2017)*, volume 10167 of *Lecture Notes in Computer Science*, pages 460–471. Springer, 2017. doi:10.1007/978-3-319-53925-6_36.

22 Duc A. Hoang, Amanj Khorramian, and Ryuhei Uehara. Shortest reconfiguration sequence for sliding tokens on spiders. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, May 27-29*, pages 262–273, Rome, Italy, 2019. Springer International Publishing. doi:10.1007/978-3-030-17402-6_22.

23 Duc A. Hoang and Ryuhei Uehara. Sliding tokens on a cactus. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:26, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2016.37.

24 Duc A. Hoang and Ryuhei Uehara. Polynomial-time algorithms for sliding tokens on cactus graphs and block graphs, 2018. arXiv:1705.00429.

25 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

26 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

27 Takehiro Ito, Marcin Kamiński, and Hirotaka Ono. Fixed-parameter tractability of token jumping on planar graphs. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation (ISAAC 2014)*, volume 8889 of *Lecture Notes in Computer Science*, pages 208–219, Jeonju, Korea, 2014. Springer International Publishing. doi:10.1007/978-3-319-13075-0_17.

28 Takehiro Ito, Hiroyuki Nooka, and Xiao Zhou. Reconfiguration of vertex covers in a graph. *IEICE Transactions on Information and Systems*, 99-D(3):598–606, 2016. doi:10.1587/TRANSINF.2015FCP0010.

29 Takehiro Ito, Hirotaka Ono, and Yota Otachi. Reconfiguration of cliques in a graph. *Discrete Applied Mathematics*, 333:43–58, 2023. doi:10.1016/j.dam.2023.01.026.

30 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

31 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Transactions on Algorithms (TALG)*, 15(1):1–19, 2018. doi:10.1145/3280825.

32 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Sebastian Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018. doi:10.3390/A11020020.

33 Noam Solomon and Shay Solomon. A generalized matching reconfiguration problem. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8*, volume 185 of *LIPIcs*, pages 57:1–57:20, Virtual Conference, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.ITCS.2021.57.

34 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

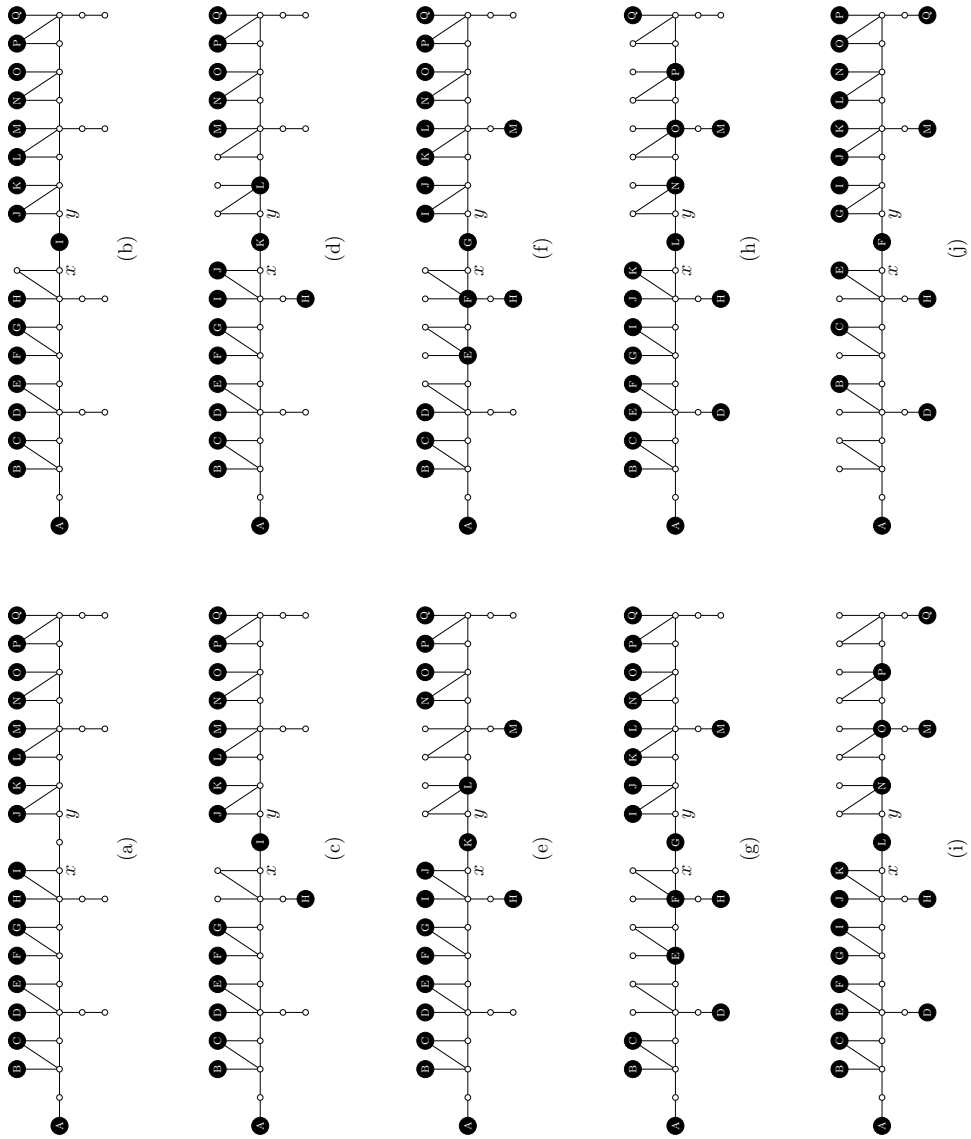


Figure 6 Token movement steps that lead to a configuration in which token A can be moved out of its block.

Fair Rent Division: New Budget and Rent Constraints

Rohith Reddy Gangam
University of California, Irvine, CA, USA

Shayan Taherijam
University of California, Irvine, CA, USA

Vijay V. Vazirani
University of California, Irvine, CA, USA

Abstract

We study the classical *rent division problem*, where n agents must allocate n indivisible rooms and split a fixed total rent R. The goal is to compute an *envy-free* (EF) allocation, where no agent prefers another agent's room and rent to their own. This problem has been extensively studied under standard assumptions, where efficient algorithms for computing EF allocations are known.

We extend this framework by introducing two practically motivated constraints: *(i)* lower and upper bounds on room rents, and *(ii)* room-specific budget for agents. We develop efficient combinatorial algorithms that either compute a feasible EF allocation or certify infeasibility.

We further design algorithms to optimize over EF allocations using natural fairness objectives such as *maximin utility*, *leximin utility*, and *minimum utility spread*. Our approach unifies both constraint types within a single algorithmic framework, advancing the applicability of fair division methods in real-world platforms such as *Spliddit*.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Rent Division, Envy-Free, Fair Division

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.32

Related Version *Full Version*: `https://arxiv.org/pdf/2510.05434`

Funding This work was supported in part by the National Science Foundation [Grant CCF-2230414].

1 Introduction

Fair rent division is a well-studied problem in economics and algorithmic game theory, and has found practical deployment through platforms such as *Spliddit* and broad public interest via the *New York Times*' interactive coverage. The classical problem involves assigning n agents to n indivisible rooms and splitting a fixed total rent R such that no agent envies another's room and rent. That is, each agent strictly prefers their own room and rent over any other's. An allocation satisfying this property is called *envy-free* (EF).

Under standard assumptions – quasi-linear utilities and no additional constraints – EF allocations are guaranteed to exist and can be computed efficiently via combinatorial or LP-based methods [13, 8]. In this work, we move beyond these assumptions and study the computation of EF allocations under two realistic extensions: bounds on individual room rents and room-specific budgets for agents.

Rent bounds on rooms arise naturally in many real-world housing arrangements. Some rooms may have contractual lower or upper limits on rent – either because landlords impose minimum rents for premium rooms or because roommates collectively agree to cap rents to maintain affordability. For instance, universities that partner with private landlords near campus must allocate housing fairly among students while honoring the minimum contractual


© Rohith Reddy Gangam, Shayan Taherijam, and Vijay V. Vazirani;
licensed under Creative Commons License CC-BY 4.0

rents specified by landlords. Such bounds capture practical and institutional constraints that cannot be represented by valuations alone, and thus warrant explicit modeling in the framework.

Our second extension considers budget constraints at the granularity of agent-room pairs. That is, each agent specifies a distinct upper bound on the rent they are willing to pay for each room. This generalizes prior models such as [10], which assume a single budget per agent. Room-specific budgets capture a variety of real-world financial considerations that valuations alone cannot express. Factors such as room-dependent utility surcharges can impose affordability constraints that vary across rooms for each agent. While valuations encode preferences, they do not reflect these heterogeneous financial limits. Incorporating agent-room-specific budgets allows us to retain expressive utilities while modeling realistic payment constraints.

We develop efficient combinatorial algorithms that either find an envy-free allocation satisfying the constraints or certify that none exists. We also provide efficient algorithms that identify EF allocations that are fair according to maximin utility, leximin utility, and minimum utility spread (both absolute and relative). Among the potentially many EF allocations, these fairness notions help identify outcomes that are more equitable and have been well studied in the literature on fair division. Finally, we give a unified algorithm that simultaneously handles both types of constraints – room rent bounds and room-specific budgets.

Organization. Section 2 reviews related work. In Section 3, we present the necessary preliminaries. Section 4 introduces an algorithm for computing envy-free allocations under bounds on room rents, along with algorithms for selecting fair allocations using maximin, leximin, and minimum spread criteria. In Section 5, we give an algorithm for handling room-specific budget constraints for agents. Finally, in Section 6, we combine both types of constraints and provide an algorithm that simultaneously satisfies them. Proofs of theorems marked † are provided in the Appendix B.

2 Related Work

The rent division problem was introduced in the economics literature by Foley [6], and further formalized by Alkan, Demange, and Gale [2]. The envy-free (EF) solution was shown to always exist under quasi-linear utilities without constraints Svensson [13], and Klijn [8] gave a strongly polynomial algorithm based on envy graphs.

More recently, Procaccia et al. [7] proposed a maximin envy-free solution and implemented it as the rent division algorithm on *Spliddit*. Their method optimizes fairness among EF solutions via linear programming. In follow-up work, Procaccia et al. [10] studied rent division with agent budgets, but assumed each agent has a single budget across rooms.

Our work generalizes this model by introducing *agent-room-specific budgets*, which better capture real-world constraints such as utility thresholds and room-specific surcharges. We provide combinatorial procedures for computing fair envy-free allocations under these constraints.

While our focus is on monetary envy-free rent division, there is significant literature on non-monetary settings. Surveys such as by Amanatidis et al. [3] review recent progress on discrete fair division problems involving indivisible goods, and notions like EF1, EFX, and

MMS. Another relevant survey by Salvatierra et al. [12] explores envy-free pricing problems in various economic settings, emphasizing the computational complexity and algorithmic aspects of maximizing seller revenue under envy-freeness constraints.

Extensions to rent division under complex settings have also been considered, including online arrivals [1], multi-apartment rent division [11] and continuous, monotone decreasing, and piecewise-linear utilities [5]. The practical relevance of this problem is evident from platforms like *Spliddit* and its popular coverage in the *New York Times* [14].

3 Preliminaries

3.1 Rent Division Problem

We consider a rent division problem involving a set of n agents $\mathcal{A} = \{1, 2, \ldots, n\}$ and a set of n rooms $\mathcal{R} = \{1, 2, \ldots, n\}$. Each room must be assigned to exactly one agent, and the room rents must be such that the total rent collected equals a fixed amount R_t. Each agent $i \in \mathcal{A}$ has a non-negative valuation $v_{ij} \in \mathbb{R}_+$ for room $j \in \mathcal{R}$, indicating their subjective worth of the room. These valuations are encoded in a valuation matrix $V = [v_{ij}] \in \mathbb{R}_+^{n \times n}$. A rent division instance is thus defined by the tuple $I = (\mathcal{A}, \mathcal{R}, V, R_t)$.

An *assignment* of rooms is a bijection $\sigma : \mathcal{A} \to \mathcal{R}$ and we will use $\sigma(i), i \in \mathcal{A}$ to denote the room assigned to agent i and abuse the notation by also using $\sigma(j), j \in \mathcal{R}$ to denote the agent assigned to room j. A *rent division* is specified by a vector $r \in \mathbb{R}^n$, where r_j denotes the rent charged for room j and $\sum_{j \in \mathcal{A}} r_j = R_t$.

Given a rent vector r, the *quasi-linear utility* of agent i for room j is defined as $u_{ij}(r) = v_{ij} - r_j$. When the rent vector is clear from context, we write u_{ij} instead of $u_{ij}(r)$. Under an assignment σ, agent i is said to *envy* agent j if $v_{i\sigma(i)} - r_{\sigma(i)} < v_{i\sigma(j)} - r_{\sigma(j)}$.

A rent allocation (σ, r) consisting of an assignment and a rent division is said to be *envy-free* (EF) if no agent envies any other, i.e.,

$$\forall i, j \in \mathcal{A}, \quad u_{i\sigma(i)} \geq u_{i\sigma(j)} \quad \text{or equivalently} \quad \forall i \in \mathcal{A}, j \in \mathcal{R} \quad u_{i\sigma(i)} \geq u_{ij}$$

3.2 Welfare-maximizing allocations

▶ **Definition 1.** *An assignment σ is said to be* welfare-maximizing *if it maximizes the total valuation, i.e., $\sum_{i \in \mathcal{A}} v_{i\sigma(i)}$.*

The following well-known result connects welfare-maximizing assignments with envy-free solutions(for reference, see [9]).

▶ **Theorem 2** (First and Second Welfare Theorems)**.** *Let (σ, r) be an envy-free (EF) allocation under instance I. Then:*

1. *σ is welfare-maximizing.*
2. *For any other welfare-maximizing assignment σ', the allocation (σ', r) is also envy-free.*

▶ **Corollary 3.** *If both (σ, r) and (σ', r) are envy-free, then each agent receives the same utility in both allocations:*

$$v_{i\sigma(i)} - r_{\sigma(i)} = v_{i\sigma'(i)} - r_{\sigma'(i)} \quad \text{for all } i \in \mathcal{A}.$$

Given the result above, the problem of finding an envy-free allocation can be naturally interpreted using a graph-theoretic framework explained below.

3.3 Envy Graph

▶ **Definition 4.** *Let $I = (\mathcal{A}, \mathcal{R}, V, R_t)$ be a rent-division instance, and let (σ, r) be an allocation of rooms and rents. Define the* envy graph *$G_E(\sigma, r) = (\mathcal{R}, E)$ on the set of rooms as follows: There is a directed edge from room i to room j if the agent assigned to room i weakly envies the agent assigned to room j under rent vector r. That is,*

$$E = \{(\sigma(i), \sigma(j)) \mid u_{i\sigma(i)} \leq u_{i\sigma(j)}\}$$

▶ **Definition 5.** *An edge $(i, j) \in E$ is called a* weak envy edge *if $u_{i\sigma(i)} = u_{i\sigma(j)}$, and a* strong envy edge *if $u_{i\sigma(i)} < u_{i\sigma(j)}$.*

This combinatorial structure has been widely utilized in the literature on envy-free rent division. In particular, Klijn [8] employed it to design a strongly polynomial-time combinatorial algorithm for computing envy-free allocations. For more details check Appendix A.1.

3.4 Model

We now extend the rent division problem to incorporate the two additional constraints of our setting. First, each agent is subject to a room-specific budget constraint, represented by a matrix $b = (b_{ij})_{(i,j)\in\mathcal{A}\times\mathcal{R}}$, where b_{ij} denotes the maximum rent agent i is willing or able to pay for room j. Second, we are given vectors $l, u \in \mathbb{R}^n$ such that each room has associated lower and upper bounds on its rent: for each room $j \in \mathcal{R}$, the rent must satisfy $l_j \leq r_j \leq u_j$.

A rent division instance with these extended constraints is defined by the tuple $I = (\mathcal{A}, \mathcal{R}, V, R_t, b, l, u)$, and in the case that there is no budget constraint or bound, we drop them from the notation. The goal is to compute an envy-free allocation that satisfies all constraints and, when possible, optimizes one of the following fairness objectives:

- *Maximin utility*: maximize the minimum utility among agents;
- *Leximin utility*: lexicographically maximize the vector of utilities;
- *Minimum absolute spread*: minimize the difference between the maximum and minimum utilities;
- *Minimum relative spread*: minimize the ratio of the maximum to minimum utilities.

We would like to note that the results in Section 4 can be obtained using Linear Programming – an optimal matching can be found separately and the conditions for envy-free rents can be written as linear constraints. This enables us to write the problem as an LP and find an envy free rent assignment that maximizes one of the above fairness objectives. The primary goal of the paper is to instead find these assignments using combinatorial means and in doing so, develop structural insights into the problem. Moreover, if the agents have room-specific bounds, these conditions can not be written as linear constraints and so, we will use these combinatorial insights to solve the problem in Section 5 and Section 6.

We first begin by analyzing the setting with agent budgets but without rent bounds.

4 Envy-free allocation with bounds on room rents

We begin by observing that the First and Second Welfare Theorems continue to hold even under these additional constraints.

▶ **Theorem 6.** † *Let (σ, r) be an envy-free allocation under instance I. Then:*

1. *σ is a welfare-maximizing assignment.*
2. *For any other welfare-maximizing assignment σ', the allocation (σ', r) is also envy-free.*

This result simplifies the problem: it suffices to first compute a welfare-maximizing assignment, and then find rents (within the given bounds) that make the allocation envy-free.

Expanding on these properties, we will describe Algorithm 7 to obtain an envy-free rent satisfying the bounds on room rents. We will then provide algorithms to obtain fair envy-free rents in Sections 4.1, and 4.2.

Let $I = (\mathcal{A}, \mathcal{R}, V, R_t, l, u)$ be a rent division instance with lower and upper bounds on the rents of each room. By Theorem 6, we may begin with any welfare-maximizing assignment and search for envy-free rents that respect the specified bounds. Algorithm 7 describes an algorithm that, given a welfare-maximizing assignment σ and an envy-free rent vector r (e.g., computed via an algorithm such as [8]), computes an envy-free allocation satisfying all rent bounds, or correctly reports that there is no solution for this instance.

We begin with an observation about the envy graph $G_E(\sigma, r)$ that is an integral part of all our algorithms. Consider an edge (i, j) in $G_E(r)$, indicating that agent i weakly envies agent j. If we decrease the rent of room j, then to maintain envy-freeness, the rent of room i must decrease by at least the same amount – otherwise, the weak envy becomes strong. Similarly, increasing the rent of room i requires increasing the rent of room j by at least as much. This implies that increasing (resp., decreasing) the rent of a room requires increasing (resp., decreasing) the rents of all rooms reachable from (resp., that can reach) it in $G_E(\sigma, r)$. for $X \subseteq \mathcal{R}$, let $R(X)$ and $R^{-1}(X)$ denote the set of rooms reachable from, and the set of rooms that can reach, some rooms in X, respectively.

Algorithm 7 operates by classifying rooms into four categories based on their current rents: L, the set of rooms priced strictly above their lower bounds; L_f, those exactly at their lower bounds; U, rooms priced strictly below their upper bounds; and U_f, those exactly at their upper bounds.

If $L \neq \varnothing$, the algorithm attempts to increase the rents of all rooms in L, along with all rooms in their reachable set $R(L)$, at unit rate. To keep the total rent fixed at R_t, it simultaneously decreases the rents of rooms not in $R(L_f) \cup R(L)$ – since decreasing the rent of a room in $R(L_f)$ would eventually require decreasing the rent of a room in L_f, which is already at its lower bound and thus infeasible.

If no such room exists, then the rent can not be altered without violating the lower bounds. the algorithm returns "No solution," as this constitutes a valid certificate that no envy-free allocation exists within the specified bounds.

The rate of decrease is carefully chosen so that the total rent remains constant. This adjustment continues until one of the following events occurs: - A room enters or leaves one of the sets L, U, L_f, or U_f, or - A new weak envy edge is introduced into the graph $G_E(\sigma, r)$. The sets and the rates are recomputed and the process is repeated until $L = \varnothing$.

Once $L = \varnothing$, a symmetric procedure is applied to reduce the rents of rooms in U (i.e., those strictly below their upper bounds), continuing until $U = \varnothing$. As shown in Theorem 7, this process produces an envy-free rent vector within bounds, if such a solution exists. Figure 5 gives a picture representation of how the algorithm works.

▶ **Theorem 7.** † *Algorithm 7 returns an envy-free rent vector satisfying the given lower and upper bounds, if such a solution exists. Otherwise, it correctly returns "No solution".*

4.1 Envy-free rents for maximin and leximin utility among agents

We first describe Algorithm 1, which computes an envy-free allocation that maximizes the minimum utility among agents, and later, extend it to obtain leximin utility. The algorithm starts with a rent vector r satisfying envy-freeness and the rent bounds, obtained via Algorithm 7. It then iteratively increases the minimum utility without violating rent bounds.

Firstly, the algorithm identifies the set M of rooms whose occupants have the minimum utility under r. It decreases the rents of rooms in M, along with all rooms in $R^{-1}(M)$, at a constant rate to preserve envy-freeness. To maintain the total rent, it increases the rents of rooms that are neither in $R^{-1}(M)$ nor in $R^{-1}(U_f)$ – i.e., rooms whose rents can be safely raised while remaining within bounds. The process continues until a new weak envy edge appears or a vertex enters or exits M, L_f, or U_f, at which point the sets and update rates are recomputed. The algorithm terminates when S_{inc}, the set of rooms eligible for rent increase, becomes empty, or when decreasing rents in M would require lowering rents of rooms in L_f, violating lower bounds. In Theorem 8, we prove that the final rent vector maximizes the minimum utility subject to envy-freeness and rent bounds. See Figure 1 for details.

Algorithm 1 Algorithm to find envy-free rents that maximize the minimum utility.

MAXIMIN-EF-RENTS$(\sigma, r, \mathcal{A}, \mathcal{R}, V, R_t, l, u)$

Input: A rent division instance with bounds on room rents, $I = (\mathcal{A}, \mathcal{R}, V, R_t, l, u)$, a welfare-maximizing assignment σ and an envy-free rent division r satisfying the bounds.

Output: An envy-free rent division r that satisfies the bounds on room rents and maximizes the minimum utility.

1. **Initialization:**
   a. $G_E(\sigma, r) :=$ Envy-Graph(σ, r)
   b. $L_f := \{j \in \mathcal{R} | r_j = l_j\}$
   c. $U_f := \{j \in \mathcal{R} | r_j = u_j\}$
   d. $M := \{j \in \mathcal{R} | u_{\sigma^{-1}(j)} = \min_{j' \in \mathcal{R}}(u_{\sigma^{-1}(j')})\}$
2. **While** $(R^{-1}(M) \cup R^{-1}(U_f)) \neq \mathcal{R}$ **do:**
   a. **If** $\boldsymbol{(R(L_f) \cap M) \neq \varnothing}$, **break**;
   b. Set $S_{dec} := R^{-1}(M)$
   c. Set $S_{inc} := \mathcal{R} \setminus (R^{-1}(M) \cup R^{-1}(U_f))$
   d. Increase rents, r_j, of all rooms in S_{inc} at unit rate;
      Decrease rents, r_j, of all rooms in S_{dec} at the rate $\frac{|S_{inc}|}{|S_{dec}|}$ simultaneously;
   e. Stop when a (weak-envy) edge is added to $G_E(r)$ or a vertex moves into or out of M, L_f, U_f.
   f. Update M, L_f, U_f and $G_E(r)$.
3. **Return:** $\boldsymbol{r}$

▶ **Theorem 8.** † *Algorithm 1 terminates and returns envy-free rents which maximize the minimum utility among agents.*

Algorithm 2 extends the Algorithm 1 to compute rents that realize the leximin utility vector. After obtaining a rent vector that maximizes the minimum utility as part of Case 1 above, the algorithm freezes the rents of all rooms in $M \cap R(L_f)$ – those with minimum utility whose rents can no longer be decreased – by adding them to both L_f and U_f. This effectively freezes their rents. It then identifies the rooms with the next lowest utility among the remaining agents and re-applies the maximin subroutine. This process continues until, like in Case 2 above, $R^{-1}(M) \cup R^{-1}(U_f) = \mathcal{R}$ – the point where no room rent can be changed without making the imputation lexicographically worse. Theorem 9 proves that the final rent vector achieves the leximin utility profile. See Figure 2 for details.

▶ **Theorem 9.** † *Algorithm 2 terminates and returns envy-free rents corresponding to the leximin utility vector.*

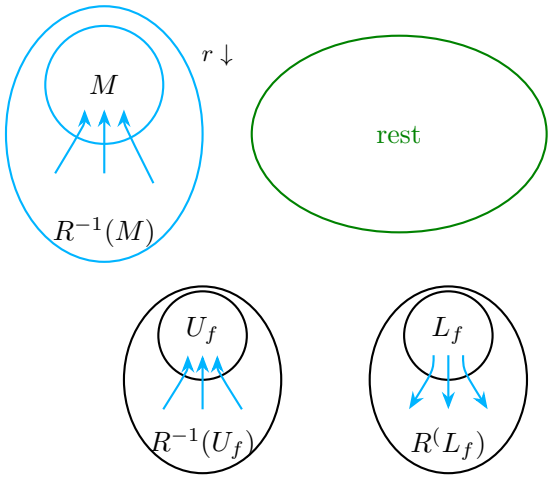


Figure 1 Figure describing Algorithm 1.

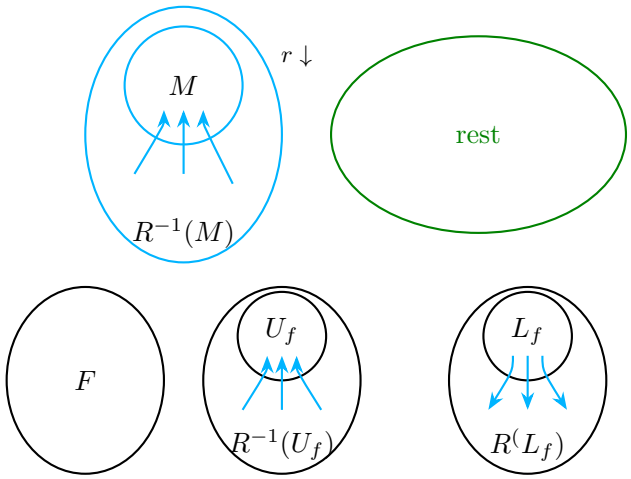


Figure 2 Figure describing Algorithm 2.

4.2 Envy-free rents that minimize the absolute and relative spread of utility among agents

Algorithm 3 computes the rents that minimize the absolute spread in utilities. It first maximizes the minimum utility using Algorithm 1 as a subroutine. Then, it freezes the rents of *these* rooms and minimizes the maximum utility among the *remaining* agents using an algorithm analogous to Algorithm 1. As Theorem 10 shows, this procedure minimizes the absolute spread in utilities. Furthermore, Theorem 11 shows that the rent vector minimizing the absolute spread also minimizes the relative spread. Hence, Algorithm 3 simultaneously minimizes both absolute and relative utility spread.

▶ **Theorem 10.** † *Algorithm 3 returns envy-free rents that minimizes the absolute spread of utility among the agents.*

▶ **Theorem 11.** † *Algorithm 3 returns envy-free rents that also minimizes the relative spread of utility among the agents.*

Algorithm 2 Algorithm to find envy-free rents corresponding to the leximin utility vector.

LEXIMIN-EF-RENTS$(\sigma, r, \mathcal{A}, \mathcal{R}, V, R_t, l, u)$

Input: A rent division instance with bounds on room rents, $I = (\mathcal{A}, \mathcal{R}, V, R_t, l, u)$, a welfare-maximizing assignment σ and an envy-free rent division r satisfying the bounds.

Output: An envy-free rent division r that satisfies the bounds on room rents and makes the utility vector leximin.

1. **Initialization:**
   a. $G_E(\sigma, r) :=$ Envy-Graph(σ, r).
   b. $L_f := \{j \in \mathcal{R} | r_j = l_j\}$
   c. $U_f := \{j \in \mathcal{R} | r_j = u_j\}$
   d. $F := \varnothing$
2. **While** $(R^{-1}(M) \cup R^{-1}(U_f)) \neq \mathcal{R}$ **do:**
   a. $M = \{j \in \mathcal{R} | u_{\sigma^{-1}(j)} = \min_{j' \in (\mathcal{R} \setminus F)}(u_{\sigma(j')})\}$
   b. **If** $\boldsymbol{(R(L_f) \cap M) \neq \varnothing}$
      i. $F := F \cup ((R(L_f) \cap M)$ *(freeze the rents)*
      ii. $L_f := L_f \cup F$
      iii. $U_f := U_f \cup F$
      iv. **continue**;
   c. Set $S_{dec} = R^{-1}(M)$
   d. Set $S_{inc} = \mathcal{R} \setminus (R^{-1}(M) \cup R^{-1}(U_f))$
   e. Increase rents, r_j, of all rooms in S_{inc} at unit rate;
      Decrease rents, r_j, of all rooms in S_{dec} at the rate $\frac{|S_{inc}|}{|S_{dec}|}$ simultaneously;
   f. Stop when a (weak-envy) edge is added to $G_E(r)$ or a vertex moves into or out of M, L_f, U_f.
   g. Update M, L_f, U_f and $G_E(r)$.
3. **Return:** $\boldsymbol{r}$

▶ Remark 12. Note that the algorithms discussed in this section change the rents at constant rate and ensure that the algorithms terminate. It is easy to compute when the next events – vertices movement to different sets or appearance of new weak-envy edges – happens. And so, we can adjust these rents discreetly to the next event. This way, all algorithms can be made to run in strongly polynomial time.

5 Envy-Free Allocation with Room-Specific Agent Budgets

In this section, we consider settings with only room-specific budgets for agents. There are no bounds on the room rents. Let $I = (\mathcal{A}, \mathcal{R}, V, R_t, b)$ be an instance of the rent division problem where $b = (b_{ij})_{(i,j) \in \mathcal{A} \times \mathcal{R}}$ specifies agent-specific budget constraints for each room. Define $EF(I)$ as the set of all envy-free allocations and $EF_b(I)$ as those that also respect the budget constraints. For a given envy-free allocation (σ, r), let $G_E(\sigma, r)$ denotes its corresponding envy graph. Since (σ, r) is envy-free, all edges in $G_E(\sigma, r)$ must be weak envy edges.

The structure of these envy graphs reveals several important properties. Notably, as established in [4], the strongly connected components of the envy graph are invariant across all allocations in $EF(I)$:

▶ **Lemma 13** ([4]). *The strongly connected components of $G_E(\sigma, r)$ remain unchanged for all $(\sigma, r) \in EF(I)$.*

This observation implies that the strongly connected components (SCCs) of the envy graph depend only on the instance $(\mathcal{A}, \mathcal{R}, V, R_t)$, and is invariant across all envy-free allocations.

Algorithm 3 Algorithm to find envy-free rents corresponding to the leximin utility vector.

MinSpread-EF-Rents$(\sigma, r, \mathcal{A}, \mathcal{R}, V, R_t, l, u)$

Input: A rent division instance with bounds on room rents, $I = (\mathcal{A}, \mathcal{R}, V, R_t, l, u)$, a welfare-maximizing assignment σ and an envy-free rent division r satisfying the bounds.

Output: An envy-free rent division r that satisfies the bounds on room rents and minimizes the absolute spread of utilities.

1. **Initialization:**
   a. $G_E(\sigma, r) :=$ Envy-Graph(σ, r)
   b. $L_f := \{j \in \mathcal{R} | r_j = l_j\}$
   c. $U_f := \{j \in \mathcal{R} | r_j = u_j\}$
   d. $M := \{j \in \mathcal{R} | u_{\sigma^{-1}(j)} = \min_{j' \in \mathcal{R}}(u_{\sigma^{-1}(j')})\}$
2. **Find maximin solution:**
   While $(R^{-1}(M) \cup R^{-1}(U_f)) \neq \mathcal{R}$ **do:**
   a. **If** $\boldsymbol{(R(L_f) \cap M) \neq \varnothing}$, **break**;
   b. Set $S_{dec} := R^{-1}(M)$
   c. Set $S_{inc} := \mathcal{R} \setminus (R^{-1}(M) \cup R^{-1}(U_f))$
   d. Increase rents, r_j, of all rooms in S_{inc} at unit rate;
   Decrease rents, r_j, of all rooms in S_{dec} at the rate $\frac{|S_{inc}|}{|S_{dec}|}$ simultaneously;
   e. Stop when a (weak-envy) edge is added to $G_E(r)$ or a vertex moves into or out of M, L_f, U_f.
   f. Update M, L_f, U_f and $G_E(r)$.
3. **Fix agents with maximin utilities:**
   a. $F := M$
   b. $L_f := L_f \cup F$
   c. $U_f := U_f \cup F$
   d. $M := \{j \in \mathcal{R} | u_{\sigma^{-1}(j)} = \min_{j' \in (\mathcal{R} \setminus F)}(u_{\sigma^{-1}(j')})\}$
4. **Find minimax solution *among the rest*:**
   While $(R(L_f) \cup R(M)) \neq \mathcal{R}$ **do:**
   a. **If** $\boldsymbol{(R^{-1}(U_f) \cap M) \neq \varnothing}$, **break**;
   b. Set $S_{dec} = \mathcal{R} \setminus (R(M) \cup R(L_f))$
   c. Set $S_{inc} = R(M)$
   d. Increase rents, r_j, of all rooms in S_{inc} at unit rate;
   Decrease rents, r_j, of all rooms in S_{dec} at the rate $\frac{|S_{inc}|}{|S_{dec}|}$ simultaneously;
   e. Stop when a (weak-envy) edge is added to $G_E(r)$ or a vertex moves into or out of M, L_f, U_f.
   f. Update M, L_f, U_f and $G_E(r)$.
5. **Return:** $\boldsymbol{r}$

In fact, the total rent R_t has no bearing on the SCC decomposition: envy-free rent vectors can be adjusted to accommodate different total rents without changing the underlying envy relations. Specifically, if $(\sigma, r) \in EF(\mathcal{A}, \mathcal{R}, V, R_t)$ is an envy-free allocation, then for any alternative total rent R'_t, we can construct a new rent vector r' defined by $r'_j = r_j + \frac{R'_t - R_t}{|\mathcal{A}|}$ for all $j \in \mathcal{R}$ such that (σ, r') is also envy-free. Since the envy graph remains unchanged, the set of SCCs is preserved.

We denote this canonical partition by $\mathcal{C}(\mathcal{A}, \mathcal{R}, V)$, which splits the room set $\mathcal{R}$ into SCCs shared across all envy-free allocations. As argued in [10], instances in which the envy graph consists of a single SCC are particularly well-structured and exhibit the following properties.

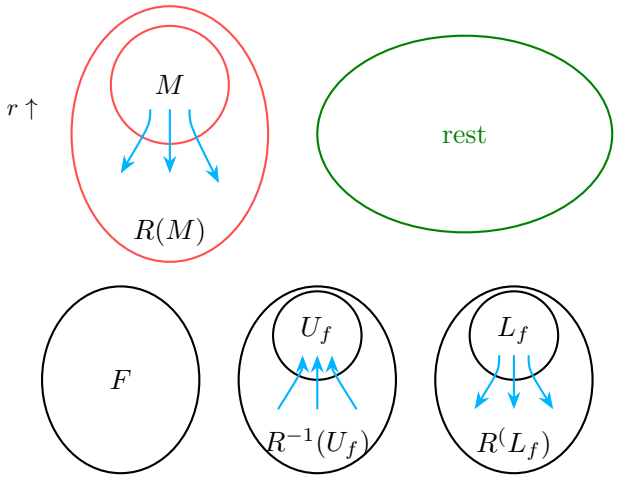


Figure 3 Figure describing Algorithm 3.

▶ **Lemma 14** ([10]). *Let $I = (\mathcal{A}, \mathcal{R}, V, R_t)$ be an instance such that $\mathcal{C}(\mathcal{A}, \mathcal{R}, V) = \{\mathcal{R}\}$. Then, for all $R_t \in \mathbb{R}$:*

1. *If $\{(\sigma, r), (\mu, r')\} \subseteq EF(\mathcal{A}, \mathcal{R}, V, R_t)$, then $r = r'$.*
2. *Each agent has the same utility among all allocations in $EF(\mathcal{A}, \mathcal{R}, V, R_t)$.*
3. *Let $(\sigma, r) \in EF(\mathcal{A}, \mathcal{R}, V, R_t)$. Then for any $R'_t \in \mathbb{R}$, and $(\mu, r') \in EF(\mathcal{A}, \mathcal{R}, V, R'_t)$, it holds that for all $a \in \mathcal{R}$, $r'_a = r_a + (R'_t - R_t)/|\mathcal{R}|$.*
4. *The rent of each room in any allocation in $EF(\mathcal{A}, \mathcal{R}, V, R_t)$ is an increasing function of R_t.*

In fact, as shown below, welfare-maximizing solutions also have nice properties corresponding to the structure of the strongly connected components.

▶ **Lemma 15** ([10]). *Let $I = (\mathcal{A}, \mathcal{R}, V, R_t)$ be a rent division instance. Then, for any two welfare-maximizing assignments σ and μ, the allocation of rooms to agents within each component $C \in \mathcal{C}(\mathcal{A}, \mathcal{R}, V)$ remains the same, i.e., $\sigma(C) = \mu(C)$.*

Consequently, in every welfare-optimal assignment, the rooms in any strongly connected component C are allocated exclusively among a fixed subset of agents. This property allows us to decompose the global problem into smaller subproblems – one for each strongly connected component – and solve them independently. The resulting component-wise solutions can then be combined to form a solution for the original instance.

Consider a strongly connected component $C \in \mathcal{C}(\mathcal{A}, \mathcal{R}, V)$. Let $\mathcal{A}(C)$ be the set of agents who are assigned rooms in C under some envy-free allocation. Define the restricted valuation and budget functions as $v_C = (v_{ij})_{i \in \mathcal{A}(C), j \in C}$ and $b_C = (b_{ij})_{i \in \mathcal{A}(C), j \in C}$. Let δ_C denote the maximum total rent with individual rationality for which an envy-free allocation respecting the budgets exists in this subproblem:

$$EF_{b_C}(\mathcal{A}(C), C, v_C, \delta_C) \neq \emptyset.$$

Continuity of utility functions guarantees the existence of such a threshold δ_C.

▶ **Lemma 16.** *Let $I = (\mathcal{A}, \mathcal{R}, V, R_t)$ be a rent division instance with agent-room-specific budgets $b = (b_{ij})_{i,j}$. Suppose that for each strongly connected component $C \in \mathcal{C}(\mathcal{A}, \mathcal{R}, V)$, there exists a valid envy-free allocation $(\sigma_C, r_C) \in EF_{b_C}(\mathcal{A}(C), C, v_C, \delta_C)$. Construct a global*

assignment σ by setting $\sigma(i) = \sigma_C(i)$ for each agent $i \in \mathcal{A}(C)$. Then, for any envy-free allocation $(\sigma', r') \in EF_b(I)$, it holds that $(\sigma, r') \in EF_b(I)$ as well.

Proof. Assume that $(\sigma', r') \in EF_b(I)$. We aim to show that $(\sigma, r') \in EF_b(I)$.

First, observe that σ is a welfare-maximizing matching, and $(\sigma', r') \in EF(I)$. By Theorem 2, it follows that $(\sigma, r') \in EF(I)$ as well. It remains to verify that (σ, r') satisfies the budget constraints.

We now prove this property for each strongly connected component (SCC) $C \in \mathcal{C}(\mathcal{A}, \mathcal{R}, V)$ separately. Let $\mathcal{A}(C)$ denote the set of agents assigned to rooms in C under σ, and define:

$$\delta'_C := \sum_{i \in \mathcal{A}(C)} r'_{\sigma'(i)}.$$

By Lemma 14, the unique rent vector r_C corresponding to σ_C (the restriction of σ to C) satisfies: $r_C = r'_C + (\delta_C - \delta'_C)\mathbf{1}$, where, δ_C denotes the maximum total rent for component C such that a solution exists in $EF_{b_C}(\mathcal{A}(C), C, v_C, \delta_C)$. Since $\delta'_C \leq \delta_C$, it follows that

$$r'_C = r_C - (\delta_C - \delta'_C)\mathbf{1} \preceq r_C.$$

Because (σ_C, r_C) satisfies the upper budget bounds by construction, and r'_C is obtained by uniformly decreasing each coordinate of r_C, the upper bounds remain satisfied. Therefore, (σ_C, r'_C) also satisfies the budget constraints. Hence, $(\sigma, r') \in EF_b(I)$. ◀

Budget–aware envy graph. For a profile (σ, r) and upper–bound vector $b = (b_{ij})_{(i,j)\in\mathcal{A}\times\mathcal{R}}$, define

$$G_b(\sigma, r) = (\mathcal{R}, E), \quad E = \left\{ (\sigma(i), \sigma(j)) \;\middle|\; u_{i\sigma(i)} \leq u_{i\sigma(j)} \text{ and } r_{\sigma(j)} < b_{i\sigma(j)} \right\}.$$

Note that, an edge from room i to room j implies that the agent in room i, can move to room j with neither decreasing their utility nor paying over their budget – representing a feasible change of rooms.

▶ **Lemma 17.** *Let $(\mathcal{A}, \mathcal{R}, V)$ be such that the entire instance forms a single strongly-connected component, $\mathcal{C}(\mathcal{A}, \mathcal{R}, V) = \{\mathcal{R}\}$, and let $b = (b_{ij})_{i,j}$ be the upper-bound matrix. Then Algorithm 4 runs in polynomial time. If (σ, r) is its output with total rent R_t, then $(\sigma, r) \in EF_b(\mathcal{A}, \mathcal{R}, V, R_t)$ and the value $\delta_{\mathcal{A}} = R_t$ is the maximum: for every $R'_t > R_t$, $EF_b(\mathcal{A}, \mathcal{R}, V, R'_t) = \emptyset$.*

Proof. First, we prove that the algorithm terminates in polynomial time. The initial envy-free (EF) solution can be computed in polynomial time using the algorithm of [8]. The main loop in line 5 has two cases. Case 1 can occur at most n^2 times, since in each iteration, the rent of a room is increased to match the budget of an agent for that room. Because rents are monotonically increasing, this adjustment cannot occur again for the same agent-room pair. As there are at most n^2 such pairs, Case 1 is executed at most n^2 times. Case 2 can occur at most n times consecutively after which, either Case 1 must be triggered or the algorithm must terminate. Therefore, the total number of iterations is bounded by a polynomial in n. Next, we claim that for each $R'_t > R_t$, $\mathsf{EF}_b(\mathcal{A}, \mathcal{R}, V, R'_t) = \emptyset$. Suppose for the sake of contradiction that there exists $R'_t > R_t$ and $(\mu, r') \in \mathsf{EF}_b(\mathcal{A}, \mathcal{R}, V, R'_t)$. Let $\Delta := (R'_t - R_t)/n$. By Lemma 14, for each $a \in \mathcal{R}$, we have $r'_a = r_a + \Delta$. Recall that $(\mu, r) \in \mathsf{EF}_b(\mathcal{A}, \mathcal{R}, V, R_t)$. Due to Corollary 3, we have that for all $i \in \mathcal{A}$,

$$v_{i\sigma(i)} - r_{\sigma(i)} = v_{i\mu(i)} - r_{\mu(i)}.$$

Algorithm 4 Maximum-rent envy-free allocation inside one strongly-connected component; $G_b(\sigma, r)$ is the budget-aware envy graph.

SCC-EF-RENTS$(\mathcal{A}, \mathcal{R}, V, b)$

Input: A rent–division instance $(\mathcal{A}, \mathcal{R}, V, b)$ with upper bounds $b = (b_{ij})_{i\in\mathcal{A}, j\in\mathcal{R}}$.

Output: The maximum feasible total rent R_t and an envy-free allocation (σ, r) that attains it.

1. **Initial EF solution:** Compute $(\sigma, r) \in EF(\mathcal{A}, \mathcal{R}, V, R_t)$ for some $R_t \in \mathbb{R}$.
2. Choose $\Delta \in \mathbb{R}$ such that

$$\big(\sigma, (r_a - \Delta)_{a\in\mathcal{R}}\big) \in EF_b(\mathcal{A}, \mathcal{R}, V, R_t - n\Delta) \quad \text{and} \quad \exists\, i \in \mathcal{A} :\ r_{\sigma(i)} = b_{i\sigma(i)}.$$

3. $r \leftarrow (r_{\sigma(i)} - \Delta)_{i\in\mathcal{A}}$.
4. $R_t \leftarrow R_t - n\Delta$.
5. **while** there is *no* $i \in \mathcal{A}$ with $r_{\sigma(i)} = b_{i\sigma(i)}$ that is *not* on a cycle of $G_b(\sigma, r)$ **do**
   a. **if** There is no agent i such that $r_i = b$ **then** *(Case 1)*
      i. $\Delta \leftarrow \min_{i\in\mathcal{A}} \big(b_{i\sigma(i)} - r_{\sigma(i)}\big)$
      ii. $r \leftarrow (r_a + \Delta)_{a\in\mathcal{R}}$
      iii. $R_t \leftarrow R_t + n\Delta$
   b. **else** *(Case 2)*
      i. Find $i \in \mathcal{A}$ with $r_{\sigma(i)} = b_{i\sigma(i)}$ that lies on a cycle C of $G_b(\sigma, r)$.
      ii. $\sigma \leftarrow$ reshuffle of σ along C
6. **end while**
7. **Return** R_t and (σ, r).

Along with

$$r_{\mu(i)} = r'_{\mu(i)} - \Delta < b_i,$$

it follows that $(\sigma(i), (\mu(i)))$ must be an edge of $G_b(\sigma, r)$ for any i. Let $\nu = \sigma^{-1} \circ \mu$, then for all $i \in \mathcal{A}$, we have

$$\sigma(i) \to \nu(\sigma(i)) \to \nu^2(\sigma(i)) \to \cdots \to \sigma(i)$$

is a cycle in $G_b(\sigma, r)$. Also, notice that the while loop only terminates when there is some $i \in \mathcal{A}$ such that this condition fails, contradicting the assumption that $r_{\sigma(i)} = b_i$. Therefore, agent i satisfies the conditions of Case 2, and the algorithm should not have terminated. ◀

Combining components. Apply Algorithm 4 independently to every SCC of $\mathcal{C}(\mathcal{A}, \mathcal{R}, V)$ and merge the results as described below.

▶ **Theorem 18.** *Algorithm 5 runs in polynomial time and returns an allocation in $EF_b(\mathcal{A}, \mathcal{R}, V, R_t)$ whenever such an allocation exists; otherwise it outputs "no solution".*

Proof. The algorithm processes each strongly connected component (SCC) by first running Algorithm 4, followed by a single execution of Algorithm 7. Since both subroutines run in polynomial time, the overall algorithm is polynomial-time. The final output is the result of Algorithm 7, and is therefore envy-free; Moreover, by construction of the instance, the solution respects the budget constraints: for each agent i assigned to room $\mu(i)$, we impose the upper bound $b_{i\mu(i)}$, ensuring that the resulting rent respects i's budget. By Lemma 16 and Theorem 6, if a feasible solution exists, it must exist for the assignment μ. Thus, if

Algorithm 5 Computing EF allocation under room-specific budgets for agents.

BudgetAware-EF$(\mathcal{A}, \mathcal{R}, V, R_t, b)$

Input: A rent division instance $(\mathcal{A}, \mathcal{R}, V, R_t, b)$ with target total rent R_t and agent-specific budgets $b = (b_{ij})$.
Output: An envy-free allocation (μ, r) satisfying the budgets and summing to R_t, or "No solution".

1. Compute an initial EF allocation $(\sigma, r) \in EF(\mathcal{A}, \mathcal{R}, V, R')$ for some R'.
2. Let $\mathcal{C} \leftarrow$ strongly connected components of $G_E(\sigma, r)$.
3. **For each** component $C \in \mathcal{C}$:
   a. $(\mu_C, r_C) \leftarrow$ SCC-EF-Rents$(\mathcal{A}_\mathcal{C}, \mathcal{R}_\mathcal{C}, V, b)$ Algorithm 4
4. Combine μ_C into a global assignment μ
5. Compute rent division r such that $(\mu, r) \in EF(\mathcal{A}, \mathcal{R}, V, R_t)$.
6. $r \leftarrow$ BoundsAware-EF-Rents$(\mu, r, \mathcal{A}, \mathcal{R}, V, R_t, -\infty, b_{i\mu(i)})$ Algorithm 7
7. **Return** (μ, r).

the algorithm returns "no solution" in line 5, no feasible EF allocation exists for any other assignment. ◀

▶ **Corollary 19.** *By substituting Algorithm 7 subroutine in line 5 of Algorithm 5 with Algorithm 1, Algorithm 2, or Algorithm 3, the algorithm computes the maximin, leximin, or minimum-spread envy-free allocations, respectively, under room-specific agent budget constraints.*

6 Envy-free allocation with bounds on room rents and room-specific agent budgets

In this section, we investigate the most general setting of the rent division problem, where we are given both room-specific lower and upper bounds on rents and agent-specific budgets for rooms. Formally, the input instance is $I = (\mathcal{A}, \mathcal{R}, V, R_t, b, l, u)$, where $b = (b_{ij})$ specifies the maximum rent agent i is willing to pay for room j, and $l = (l_j)$ and $u = (u_j)$ define lower and upper bounds on each room's rent.

▶ **Theorem 20.** *There exists a polynomial-time algorithm that computes an envy-free allocation (σ, r) satisfying both the budget constraints b and the room rent bounds l, u. Furthermore, the algorithm can be modified to return an envy-free allocation optimizing any of the following objectives among all feasible envy-free allocations: (i) lexicographically maximal utilities (leximin), (ii) lexicographically minimal utilities (leximax), (iii) maximum among minimum utilities (maximin), (iv) minimum utility spread (min-spread).*

Proof. Run Algorithm 5 on $(\mathcal{A}, \mathcal{R}, V, b, R_t)$ to computes an envy-free allocation (σ, r') that satisfies the agent-specific budget constraints b but may violate room rent bounds l and u.

To ensure that the final rents also respect the room bounds, we define new upper bounds $\tilde{u}_j := \min\{u_j, b_{\sigma^{-1}(j)\,j}\}$ for each room j. These bounds guarantee that the rent paid for room j will not exceed either the agent's budget for room j or the specified room cap.

Next, we use Algorithm 7 to compute a new rent vector r satisfying the bounds $l_j \le r_j \le \tilde{u}_j$, while preserving envy-freeness with respect to the fixed assignment σ. This ensures that the final allocation (σ, r) satisfies both types of constraints.

Algorithm 6 Combined EF allocation with budgets and rent bounds.

BudgetAndBoundsAware-EF-Rents$(\mathcal{A}, \mathcal{R}, V, R_t, b, l, u)$
Input: Instance $I = (\mathcal{A}, \mathcal{R}, V, R_t, b, l, u)$
Output: Envy-free allocation (σ, r) satisfying all constraints
1. $\sigma \leftarrow$ BudgetAware-EF$(\mathcal{A}, \mathcal{R}, V, R_t, b)$ (Algorithm 5)
2. **for each** room $j \in \mathcal{R}$:
   - Let $i = \sigma^{-1}(j)$ be the agent assigned to room j
   - Set $\tilde{u}_j \leftarrow \min\{u_j,\ b_{ij}\}$
3. $r \leftarrow$ BoundsAware-EF-Rents$(\sigma, \mathcal{A}, \mathcal{R}, V, R_t, l, \tilde{u})$ (Algorithm 7)
4. **Return** (σ, r)

Since each step uses polynomial-time subroutines and the number of rooms is n, the overall algorithm runs in polynomial time.

For the optimization objectives: Once the assignment σ is fixed, the problem becomes finding an envy-free solution with a fixed assignment and bounds on rents $l_j \le r_j \le \tilde{u}_j$, that in Section 4 we have provided solutions for different problems in this setting such as: *maximin* can be solved via Algorithm 1, *leximin* can be solved via Algorithm 2, and *min-spread* can be solved via Algorithm 3.
Note that if there exists an envy-free allocation (μ, r') that satisfies both types of constraints, then by Lemma 17 and Theorem 6, the allocation (σ, r') must also satisfy both types of constraints. Therefore, if the algorithm outputs "no solution" for the assignment σ, it implies that no feasible envy-free solution exists for any assignment. ◀

▶ Remark 21. Given a fixed assignment of agents to rooms, all envy-free, rent bounds, and budget constraints can be written as linear constraints. Consequently, once we have found the matching, the fair envy-free rents can be computed under that assignment and can also be computed via linear programming, refer to Procaccia et al. [10] for more details.

References

1 Dimitar Aleksandrov and Toby Walsh. Online fair division: Survey and directions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.
2 Mehmet Alkan, Gabrielle Demange, and David Gale. Fair allocation of indivisible goods and money. *Journal of Economic Theory*, 55(2):321–333, 1991.
3 Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A. Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artificial Intelligence*, 322:103965, 2023. doi:10.1016/j.artint.2023.103965.
4 Tommy Andersson, Lars Ehlers, and Lars-Gunnar Svensson. Budget balance, fairness, and minimal manipulability. *Theoretical Economics*, 9(3):753–777, 2014. doi:10.3982/TE1346.
5 Eshwar Ram Arunachaleswaran, Siddharth Barman, and Nidhi Rathi. Fully polynomial-time approximation schemes for fair rent division. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1994–2013, USA, 2019. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975482.121.
6 Duncan Foley. Resource allocation and the public sector. *Yale Economic Essays*, 7(1):45–98, 1967.
7 Kobi Gal, Moshe Mash, Ariel D Procaccia, and Yair Zick. Which is the fairest (rent division) of them all? *ACM Transactions on Economics and Computation (TEAC)*, 4(4):1–27, 2016.

8 Flip Klijn. An algorithm for envy-free allocations in an economy with indivisible objects and money. *Social Choice and Welfare*, 17(2):201–215, 2000. doi:10.1007/s003550050015.

9 Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

10 Ariel Procaccia, Rodrigo Velez, and Dingli Yu. Fair rent division on a budget. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. doi:10.1609/aaai.v32i1.11465.

11 Ariel D Procaccia, Benjamin Schiffer, and Shirley Zhang. Multi-apartment rent division. *arXiv preprint arXiv:2403.08051*, 2024.

12 Marcos Salvatierra, Juan G. Colonna, Mario Salvatierra Jr., and Alcides de C. Amorim Neto. On Complexity Classes of Envy-Free Pricing Problems: A Short Survey. *SN Operations Research Forum*, 5(4):1–25, December 2024. doi:10.1007/s43069-024-00373-1.

13 Lars-Gunnar Svensson. Large indivisibles: An analysis with respect to price equilibrium and fairness. *Econometrica*, 51(4):939–954, 1983.

14 The New York Times. To divide the rent, start with a triangle. https://www.nytimes.com/interactive/2014/science/rent-division-calculator.html, 2014. Accessed June 2025.

A Appendix

A.1 Klijn's Algorithm

Given an instance of the rent division problem $I = (\mathcal{A}, \mathcal{R}, V, R_t)$, begin with an arbitrary allocation of rooms and rents, (σ, r), and construct the corresponding envy graph $G_E(\sigma, r)$. If the graph contains a cycle with at least one strong envy edge, then reassign(/permute) agents along the cycle so that each agent moves to the room they weakly prefer within the cycle.

If there exists a strong envy edge (i, j) that is not part of any cycle, proceed as follows. Let $R^{-1}(i)$ denote the set of rooms from which there is a directed path to room i (the inverse reachability set), and let $R(j)$ denote the set of rooms reachable from room j (the reachability set). Since (i, j) is not part of any cycle, these two sets are disjoint.

Now, increase the rents of all rooms in $R^{-1}(i)$ at unit rate, and decrease the rents of all rooms in $R(j)$ at rate $\frac{|R^{-1}(i)|}{|R(j)|}$. This adjustment preserves the total rent. As rents evolve under this scheme, one of two things will happen: either the strong edge (i, j) becomes weak, or new weak edges appear that create a cycle containing at least one strong edge or increase the size of $R^{-1}(i) \cup R(j)$.

In either case, Klijn [8] proves that this iterative procedure makes progress by an elimination of a strong edge and hence the algorithm terminates in strongly polynomial time. For a full description and analysis, see [8].

▶ **Observation 22.** *For any rent vector r, the envy graph corresponding to a welfare-maximizing assignment contains no cycles with strong envy edges. Consequently, if Klijn's algorithm [8] is initialized with a welfare-maximizing assignment, the matching remains fixed throughout, and only the rents need to be adjusted to eliminate all strong envy edges and achieve envy-freeness.*

▶ **Example 23** (Leximin vs. Min-Spread)**.** Consider the following rent division instance with four agents and four rooms. The valuation matrix $V \in \mathbb{R}_+^{4\times 4}$ and its unique maximum weight matching is shown below:

$$V = \begin{bmatrix} 20 & 0 & 20 & 0 \\ 0 & 19 & 0 & 0 \\ 5 & 0 & 5 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

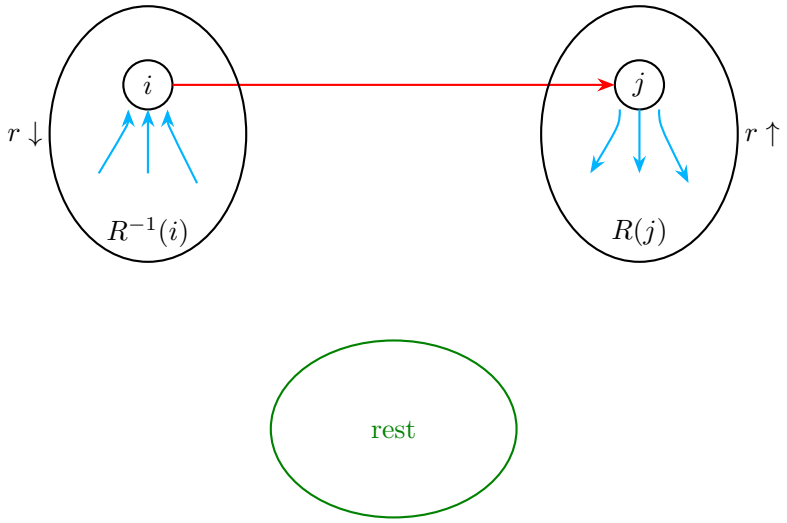


Figure 4 Figure describing algorithm in Appendix A.1.

The total rent is 4 and each room j must be priced within the bounds $[l_j, u_j]$ shown below:

$$[l_1, u_1] = [0, 2], \quad [l_2, u_2] = [0, 2], \quad [l_3, u_3] = [0, 2], \quad [l_4, u_4] = [2, 2]$$

Two rent vectors (EF under the same assignment) are:

Leximin: $r = (0, 2, 0, 2) \quad \Rightarrow \quad u = (20, 8, 5, 0)$

Min-Spread: $r = (1, 0, 1, 2) \quad \Rightarrow \quad u = (19, 10, 4, 0)$

Both are envy-free and satisfy the rent bounds, but they differ in their fairness objectives:

- The first maximizes the lexicographically minimum utility vector (**leximin**).
- The second minimizes the difference between maximum and minimum utilities (**min-spread**).

This illustrates that the leximin solution does not always yield the minimum spread.

B Omitted Proofs

▶ **Theorem 6.** † *Let (σ, r) be an envy-free allocation under instance I. Then:*

1. *σ is a welfare-maximizing assignment.*
2. *For any other welfare-maximizing assignment σ', the allocation (σ', r) is also envy-free.*

Proof. Let σ' be any welfare-maximizing assignment. Since (σ, r) is envy-free, each agent i weakly prefers their allocation under σ to that under σ':

$$v_{i\sigma(i)} - r_{\sigma(i)} \geq v_{i\sigma'(i)} - r_{\sigma'(i)} \quad \text{for all } i \in \mathcal{A}.$$

Summing over all agents and rearranging,

$$\sum_{i \in \mathcal{A}} v_{i\sigma(i)} - \sum_{j \in \mathcal{R}} r_j \geq \sum_{i \in \mathcal{A}} v_{i\sigma'(i)} - \sum_{j \in \mathcal{R}} r_j.$$

Since the total rent $\sum_j r_j = R_t$ is fixed, we get: $\sum_{i\in\mathcal{A}} v_{i\sigma(i)} \geq \sum_{i\in\mathcal{A}} v_{i\sigma'(i)}$. But σ' is welfare-maximizing, so this inequality must be tight. Hence, σ is also welfare-maximizing, and the initial inequalities for each agent must also be equalities, i.e., for each $i \in \mathcal{A}$,

$$v_{i\sigma(i)} - r_{\sigma(i)} = v_{i\sigma'(i)} - r_{\sigma'(i)} \geq v_{ij} - r_j \quad \forall j \in \mathcal{R}.$$

Thus, (σ', r) is also envy-free. It also satisfies the bound constraints because (σ, r) does, and we have the same renting here as well. ◀

▶ **Theorem 7.** † *Algorithm 7 returns an envy-free rent vector satisfying the given lower and upper bounds, if such a solution exists. Otherwise, it correctly returns "No solution".*

Proof. We will prove the above theorem in multiple parts.

Algorithm 7 Algorithm to find envy-free rents satisfying lower and upper bounds on room rents; $R(X)$ and $R^{-1}(X)$ correspond to the set of vertices that are reachable from, and the set of vertices that can reach some vertex of X in the graph $G_E(r)$ respectively.

BoundsAware-EF-Rents$(\sigma, r, \mathcal{A}, \mathcal{R}, V, R_t, l, u)$

Input: A rent division instance with bounds on room rents, $I = (\mathcal{A}, \mathcal{R}, V, R_t, l, u)$, a welfare-maximizing assignment σ and some envy-free rent division r.

Output: An envy-free rent division r that satisfies the bounds on room rents.

1. **Initialization:**
   a. $G_E(\sigma, r) :=$ Envy-Graph(σ, r)
   b. $L := \{j \in \mathcal{R} | r_j < l_j\}$
   c. $U := \{j \in \mathcal{R} | r_j > u_j\}$
   d. $L_f := \{j \in \mathcal{R} | r_j = l_j\}$
   e. $U_f := \{j \in \mathcal{R} | r_j = u_j\}$
2. **While** $L \cup U \neq \varnothing$ **do:**
   a. **If** $U \cap \{R(L_F) \cup R(L)\} \neq \varnothing$, **return** "No solution";
   b. **If** $L \cap \{R^{-1}(U_F) \cup R^{-1}(U)\} \neq \varnothing$, **return** "No solution";
   c. Set $S_{inc} := \begin{cases} R(L) & \text{if } L \neq \varnothing \\ \mathcal{R} \setminus \{R^{-1}(U_F) \cup R^{-1}(U)\} & \text{otherwise} \end{cases}$
   d. Set $S_{dec} := \begin{cases} R^{-1}(U) & \text{if } U \neq \varnothing \\ \mathcal{R} \setminus \{R(L_F) \cup R(L)\} & \text{otherwise} \end{cases}$
   e. Increase rents, r_j, of all rooms in S_{inc} at unit rate;
   Decrease rents, r_j, of all rooms in S_{dec} at the rate $\frac{|S_{inc}|}{|S_{dec}|}$ simultaneously;
   f. Stop when a (weak-envy) edge is added to $G_E(r)$ or a vertex moves into or out of L, U, L_f, U_f.
   g. Update L, U, L_f, U_f and $G_E(r)$.
3. **Return:** r

▶ **Lemma 24.** *If Algorithm 7 returns "No solution", then there exists no envy-free rents that satisfies the given bounds.*

Proof. Assume that the rents computed by the algorithm before returning "No solution" is denoted by r, and let r' be an envy-free (EF) solution that satisfies the bounds.

When the algorithm returns "No solution", one of the following two events must have occurred:

1. $U \cap (R(L_F) \cup R(L)) \neq \varnothing$

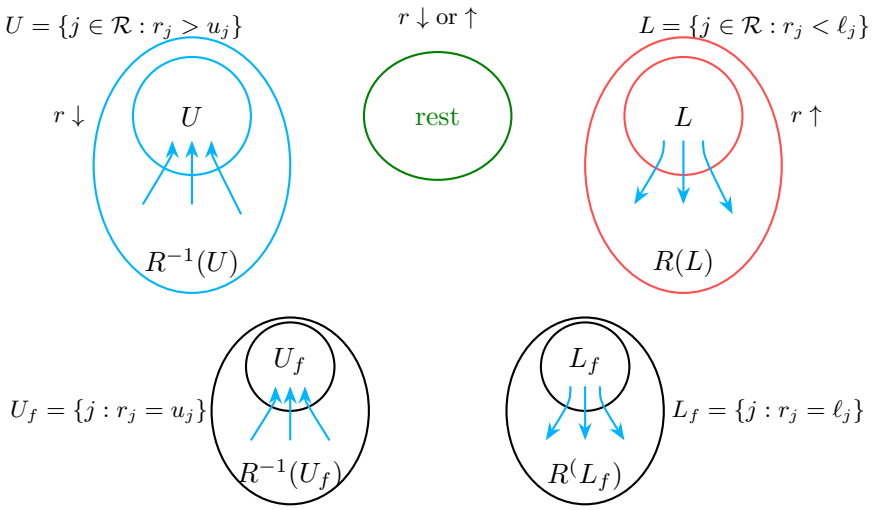


Figure 5 Figure describing Algorithm 7.

2. $L \cap \left(R^{-1}(U_F) \cup R^{-1}(U)\right) \neq \varnothing$

In case (1), there exists a room $j \in L_F \cup L$ and a room $k \in U$ such that there is a path from j to k in the weak envy graph under rent r. Let this path be $j, h_1, h_2, \ldots, h_t, k$.

Since r' is an EF rent that satisfies the bounds, we have $r'_k < r_k$. Because there is a weak envy edge from h_t to k, it follows that $r'_{h_t} < r_{h_t}$. Continuing this reasoning along the path, we obtain $r'_j < r_j$.

However, since $j \in L_F \cup L$, we know that $r_j \leq \ell_j$, and thus $r'_j < \ell_j$, which contradicts our assumption that r' satisfies the bounds.

An analogous argument applies for case (2), completing the proof. ◀

▶ **Lemma 25.** *If Algorithm 7 returns a rent r, then r is an envy-free rent satisfying the bounds on rents.*

Proof. The algorithm begins with an envy-free rent vector r that satisfies the total rent constraint. Throughout the while loop, the rents are adjusted in a way that preserves the total rent. Whenever a new weak-envy edge appears, the relevant sets are updated to ensure that no strong envy is introduced. The algorithm terminates and returns r only when the while loop condition fails – that is, when all room rents lie within their respective bounds. Therefore, the returned rent vector r is envy-free, respects the total rent constraint, and satisfies the given lower and upper bounds. ◀

▶ **Lemma 26.** *The Algorithm 7 terminates.*

Proof. We show that after $O(n)$ iterations of the while loop, $|U \cup L|$ will decrease. Let's consider two cases:

1. $U \neq \emptyset$ and $L \neq \emptyset$
2. $U = \emptyset$ or $L = \emptyset$

In the first case, we decrease the rents of $R^{-1}(U)$ and increase the rent of $R(L)$, so new edges will be coming into $R^{-1}(U)$ or going out of $R(L)$. Whenever a new edge is introduced, it will increase $|R^{-1}(U) \cup R(L)|$ or move a vertex from U to U_f or from L to L_f. Since we only have n vertices, it will decrease $|L \cup U|$ after at most n iterations of the while loop.

For the second case, let's say $U \neq \emptyset$ and $L = \emptyset$. The algorithm will decrease the rent of $R^{-1}(U)$ and increase the rent of $\mathcal{R} \setminus (R^{-1}(U_f) \cup R^{-1}(U))$. Let's first do these changes at a very small rate, so we only move all L_f out of it, so let's assume that $L_f = \emptyset$. After each step of the algorithm, either a new edge will come out from the $\mathcal{R} \setminus (R^{-1}(U_f) \cup R^{-1}(U))$ or a new edge will go in the $R^{-1}(U)$. Hence, either $|R^{-1}(U)|$ or $|(R^{-1}(U_f) \cup R^{-1}(U))|$ increases, which again can only happen at most n times after which $|L \cup R|$ must decrease. As $|U \cup L|$ can only decrease n times, the algorithm will terminate in at most n^2 iterations of the while loop. ◀

◀

▶ **Theorem 8.** † *Algorithm 1 terminates and returns envy-free rents which maximize the minimum utility among agents.*

Proof. Assume the algorithm terminates with output r with σ as the associated matching. Suppose, for contradiction, that (σ, r) does not maximize the minimum utility, and there exists another envy-free allocation (μ, r') that does, achieving a higher minimum utility.

By Theorem 2, since both σ and μ are welfare-maximizing matchings, (σ, r') is also envy-free. Then, by Corollary 3, the utilities of all agents under (μ, r') and (σ, r') are identical. Hence, (σ, r') achieves the same (higher) minimum utility as (μ, r'), and so the minimum utility under (σ, r') is strictly higher than under (σ, r).

Now consider the set M of rooms whose occupants have the minimum utility at the end of the algorithm. Since r' achieves a higher minimum utility, every agent assigned to a room in M must receive strictly more utility under r' than under r. Note that the algorithm terminates under one of two conditions:

1. $R(L_f) \cap M \neq \varnothing$, or
2. $R^{-1}(M) \cup R^{-1}(U_f) = \mathcal{R}$.

Case 1: $R(L_f) \cap M \neq \varnothing$. There exists a room $j \in L_f$ and a room $k \in M$ such that there is a directed path from j to k in the envy graph $G_E(\sigma, r)$:

$$j = h_1 \to h_2 \to \cdots \to h_t = k.$$

Since (σ, r') is envy-free and provides higher utility to the agent in room k, it must be that $r'_k < r_k$. To avoid introducing strong envy from the agent in h_{t-1} to k, we must have $r'_{h_{t-1}} < r_{h_{t-1}}$. Continuing this argument along the path, we eventually get $r'_j < r_j$. But since $j \in L_f$, we have $r_j = l_j$, and any further decrease would violate the lower bound. Thus, r'_j is infeasible – a contradiction.

Case 2: $R^{-1}(M) \cup R^{-1}(U_f) = \mathcal{R}$. This implies that every agent can reach some room in $M \cup U_f$ via a directed path in $G_E(\sigma, r)$. Using the same logic as in Case 1, we conclude that $r'_j \leq r_j$ for all j, and $r'_j < r_j$ for all $j \in M$. But this contradicts the fact that the total rent is same in both r and r' , as we have strictly decreased the rent of some rooms and weakly decreased the rest.

Therefore, our initial assumption is false, and (σ, r) must be a maximin utility solution.

To prove that the algorithm terminates, we argue that $|R^{-1}(M) \cup R^{-1}(U_f)|$ increases after at most n iterations of the while loop. Note that when a new edge is created in the algorithm, it should either go into $R^{-1}(M)$ or out of $\mathcal{R} \setminus (R^{-1}(M) \cup R^{-1}(U_f))$. Either way, we will have a new edge going into $(R^{-1}(M) \cup R^{-1}(U_f))$ and so, the set of vertices that can reach $(R^{-1}(M)$ or $R^{-1}(U_f)$ only increase. Since this can happen only $O(n)$ times, the algorithm terminates within $O(n)$ iterations of the while loop. ◀

▶ **Theorem 9.** † *Algorithm 2 terminates and returns envy-free rents corresponding to the leximin utility vector.*

Proof. Assume the algorithm starts with matching σ and terminates with the output r but say $r'(\neq r)$ is the leximin utility rents. Let's say iteration l is the first time the algorithm fixes the rent of a room in r to a value that is different to that of r' by either adding it to F or exiting the while loop. Note that all the rooms whose rents match in r and r' and have been frozen previously are in F. Like in Theorem 8 are two cases how iteration l ends. The proofs of both these cases very much mimic Theorem 8 and we explain them briefly.

Case 1: $R(L_f) \cap M \neq \varnothing$. There exists a room $j \in L_f$ and a room $k \in M$ such that there is a directed path from j to k in the envy graph $G_E(\sigma, r)$: $j = h_1 \to h_2 \to \cdots \to h_t = k$. Since (σ, r') is envy-free and provides higher utility to the agent in room k, it must be that $r'_k < r_k$. To avoid introducing strong envy from the agent in h_{t-1} to k, we must have $r'_{h_{t-1}} < r_{h_{t-1}}$. Continuing this argument along the path, we eventually get $r'_j < r_j$. But since $j \in L_f$, we have $r_j = l_j$ or $r_j = r'_j$, and any further decrease would violate either one of these necessary conditions giving a contradiction.

Case 2: $R^{-1}(M) \cup R^{-1}(U_f) = \mathcal{R}$. This implies that every agent can reach some room in $M \cup U_f$ via a directed path in $G_E(\sigma, r)$. Using the same logic as in Case 1, we conclude that $r'_j \leq r_j$ for all j, and $r'_j < r_j$ for all $j \in M$. But this contradicts the fact that the total rent is the same in both r and r' , as we have strictly decreased the rent of some rooms and weakly decreased the rest. Since neither case is possible, (σ, r) is the leximin envy-free solution.

Proof of termination of this algorithm proceeds in the same way as Algorithm 1.

To prove that the algorithm terminates we argue that $|R^{-1}(M) \cup R^{-1}(U_f)|$ increases after at most n iterations of the while loop before $|F|$ increases. Note that when a new edge is created in the algorithm, it should either go into $R^{-1}(M)$ or out of $\mathcal{R} \setminus (R^{-1}(M) \cup R^{-1}(U_f))$. Either way, we will have a new edge going into $(R^{-1}(M) \cup R^{-1}(U_f))$ and so, The set of vertices that can reach $(R^{-1}(M)$ or $R^{-1}(U_f)$ only increase. Since, this can happen only $O(n)$ times, the size of $|F|$ increases within $O(n)$ iterations of the while loop. And since $|F|$ can only increase from 0 to n, the algorithm terminates in $O(n^2)$ iterations of the while loop. ◀

▶ **Theorem 10.** † *Algorithm 3 returns envy-free rents that minimizes the absolute spread of utility among the agents.*

Proof. The argument to show that the algorithm terminates is very similar to Algorithm 1 and Algorithm 2 and is omitted for brevity.

First, note that r is a maximin solution. Assume the algorithm starts with matching σ and terminates with the output r. Note that r is a maximin utility rent vector – after computing the maximin solution, the algorithm freezes the rents of these rooms and only changes the rents of other rooms. Let $r'(\neq r)$ be envy-free rents satisfying the budgets with a lesser spread than r. AS r is a maximin utility rent vector, the minimum utility of agents in r' cannot be higher than in r. And since r' has a smaller spread than r, the maximum utility of an agent in r' must be less than that in r.

Now consider the set M of rooms whose occupants have the maximum utility(under r) at the end of the algorithm. Every agent assigned to a room in M must receive less utility under r' than under r. Note that the algorithm terminates under one of two conditions:

1. $R^{-1}(U_f) \cap M \neq \varnothing$, or
2. $R(M) \cup R(L_f) = \mathcal{R}$.

Case 1: $R^{-1}(U_f) \cap M \neq \varnothing$. There exists a room $k \in U_f$ and a room $j \in M$ such that there is a directed path from j to k in the envy graph $G_E(\sigma, r)$:

$$j = h_1 \to h_2 \to \cdots \to h_t = k.$$

Since (σ, r') is envy-free and provides lower utility to the agents in M, it must be that $r'_j > r_j$, i.e., rent of room j must go up from r to r'. To avoid introducing strong envy from the agent in j to h_1, the rent of h_1 should go up by at least that of j. So, we have $r'_{h_1} - r_{h_1} \geq r'_j - r_j > 0$. Continuing this argument along the path, we get $r'_k - r_k \geq r'_j - r_j > 0$.

But since $k \in U_f$, we have either $r_k = u_k$ or k is a room with minimum utility in r. If $r_k = u_k$, then $r'_k > r_k = u_k$ making r' infeasible. If k is a room with minimum utility in r, then $r'_k - r_k \geq r'_j - r_j \Leftrightarrow u'_\sigma(j) - u'_\sigma(k) \geq u_\sigma(j) - u_\sigma(k)$. Note that $u_\sigma(j) - u_\sigma(k)$ is just the spread of u as $j \in M$ and k are the rooms with maximum and minimum utility respectively.

And so, r' is either infeasible or has at most the spread of r – a contradiction.

Case 2: $R(M) \cup R(L_f) = \mathcal{R}$. This implies that every room is reachable from some room in $M \cup L_f$ via a directed path in $G_E(\sigma, r)$. Using the same logic as in Case 1 for every of these paths, we get that $\forall k \in \mathcal{R}, j \in M, r'_k - r_k \geq r'_j - r_j$, and $r'_j > r_j$ for all $j \in M$. But this contradicts the fact that the total rent is same in both r and r' , as we have strictly increased the rent of all rooms from r to r'.

Therefore, our initial assumption is false, and (σ, r) must be a min-spread utility envy-free rent solution. ◀

▶ **Theorem 11.** † *Algorithm 3 returns envy-free rents that also minimizes the relative spread of utility among the agents.*

Proof. By Theorem 10 we know the algorithm returns envy-free rents that minimize the absolute spread, that is also the maximin solution. We claim that these two imply minimum relative spread utility.

Assume that (σ, r) is the output of the Algorithm 3, and agents a, b have minimum and maximum utilities, respectively. Let's say (σ', r') is the min relative spread utility solution, and agents a', b' have minimum and maximum utilities, respectively.

By abusing the notation let's call

$$u_a = u_{a\sigma(a)(r)}, \quad u_b = u_{b\sigma(b)(r)}, \quad u_{a'} = u_{a'\sigma'(a')(r')}, \quad u_{b'} = u_{b'\sigma'(b')(r')}$$

We know (σ, r) has maximin utility and so, $u_a \geq u_{a'}$. Since, (σ, r) has min-spread utility, we have $u_b - u_a \leq u_{b'} - u_{a'}$. Therefore, $\frac{u_b - u_a}{u_a} \leq \frac{u_{b'} - u_{a'}}{u_{a'}}$

$$\frac{u_b - u_a}{u_a} \leq \frac{u_{b'} - u_{a'}}{u_{a'}} \Leftrightarrow \frac{u_b}{u_a} \leq \frac{u_{b'}}{u_{a'}}$$

So, the relative spread of the maximum and minimum utility agents under vector r is always at most that of the same agents under vector r'. Since, r' has the minimum relative spread, so does r, completing the proof. ◀

A Zone-Based Algorithm for Timed Parity Games

Gilles Geeraerts
Université libre de Bruxelles, Belgium

Frédéric Herbreteau
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, 33400 Talence, France

Jean-François Raskin
Université libre de Bruxelles, Belgium

Alexis Reynouard
Université libre de Bruxelles, Belgium

Abstract

This paper revisits timed games by building upon the semantics introduced in "The Element of Surprise in Timed Games" [9]. We introduce some modifications to this semantics for two primary reasons: firstly, we recognize instances where the original semantics appears counterintuitive in the context of controller synthesis; secondly, we present methods to develop efficient zone-based algorithms. Our algorithm successfully addresses timed parity games, and we have implemented it using UppAal's zone library. This prototype effectively demonstrates the feasibility of a zone-based algorithm for parity objectives and a rich semantics for timed interactions between the players.

2012 ACM Subject Classification Software and its engineering → Formal methods

Keywords and phrases Timed Parity Games, Realtime Controller Synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.33

Related Version *Extended Version*: https://arxiv.org/abs/2510.07361

1 Introduction

Timed automata [1] are a well-established formalism for modeling real-time systems. Their semantics captures the continuous evolution of time along with discrete transitions, making them particularly suitable for specifying and verifying time-critical behaviors. A crucial aspect in their analysis is the treatment of infinite executions: runs in which time converges are usually deemed unrealistic, as they imply infinitely many actions occurring within a finite amount of time, a phenomenon called *zenoness*.

Timed games extend the model of timed automata to a two-player setting, where one player represents the system under design and the other represents its environment. Defining the semantics of such games is challenging. A central issue is that one player might attempt to win by taking actions increasingly closer in time, effectively blocking the opponent's ability to respond. However, such strategies rely on blocking the progress of time and are thus physically implausible and should not be permitted.

The paper "The Element of Surprise in Timed Games" [9] proposes an elegant solution to this problem by introducing a timed game model that preserves both the element of surprise (players are allowed to play quickly in order to overtake their adversary), and the need for time divergence. In this model, players independently and simultaneously choose both an action and a delay before executing it. The action with the shortest delay is taken, introducing inherent nondeterminism in the case of equal delays. To ensure physical realism, the semantics prevents a player from winning by stopping time and introduces symmetric winning conditions applicable to all ω-regular objectives. The authors show that the ability to surprise the opponent increases the expressive power of the model and demonstrate that memory strategies may be necessary even for reachability objectives. Although the semantics


© Gilles Geeraerts, Frédéric Herbreteau, Jean-François Raskin, and Alexis Reynouard;
licensed under Creative Commons License CC-BY 4.0

is shown to be decidable via region-based abstraction, no implementation currently exists. In particular, this semantics requires solving parity games over timed structures, a task for which no practical implementation is available. Existing tools only support simpler objectives such as safety and reachability.

For simpler games like safety and reachability games, the paper [6] presents a fully symbolic zone-based on-the-fly algorithm for solving reachability and safety timed games. This approach is based on a forward exploration of the state space using zones and supports early termination. The algorithm is efficient in practice and was evaluated experimentally, but it relies on a simpler semantics that does not include the surprise mechanism or handle general ω-regular objectives.

Our objective in this work is to provide an efficient implementation of a semantics closely related to that introduced by de Alfaro et al. [9], which preserves the notion of *surprise* in real-time interactions. This will be the first efficient implementation of synthesis for parity objective in a timed two-player context. To this end, we slightly adapt the original semantics with two goals: to make it amenable to a symbolic zone-based implementation; and to handle the controller synthesis problem in a natural way, while retaining the essential features of the original model. We then develop a symbolic algorithm for solving parity games under this expressive timed semantics. Our method builds on ideas from Zielonka's recursive algorithm for parity games on finite graphs [14]. To the best of our knowledge, this is the *first implementation* of an algorithm for solving *timed parity games*, marking a significant advance in the analysis and synthesis of real-time systems with complex winning conditions.

Related works

Above, we discussed the foundational contributions of de Alfaro et al. [9] and Cassez et al. [6], which address the semantics and algorithmic analysis of timed games respectively. Beyond these, several studies have explored related challenges in timed games.

Cassez et al. [7] examine various formulations of control problems for timed and hybrid systems, highlighting the limitations of dense-time controllers and advocating for more implementable discrete-time approaches. Their solution is obtained through a discretization of the timed semantics which is avoided by [9]. At the initial phase of developing real-time systems, discretization is typically deemed less suitable than continuous time semantics that are more abstract. Brihaye et al. [5] focuses on the minimum-time reachability problem in timed games, presenting methods to compute the least time required for a player to reach a target location against all possible strategies of the opponent. In a case study paper, Cassez et al. [8] demonstrate the application of synthesis tools like UppAal-TiGa for the automatic generation of robust and near-optimal controllers in an industrial application.

Addressing the implementability of timed models, De Wulf et al. [13] introduce the "Almost ASAP" semantics, a relaxation of the traditional ASAP semantics, designed to better reflect the limitations of physical systems, especially when implementing precisely timed strategies on hardware with digital clocks.

Bouyer et al. [3] investigate the value problem in weighted timed games, proving its undecidability in the general case [4], but also providing approximation algorithms for certain subclasses.

Realizability problems for real-time logics have been explored in [11, 10], where it was shown that only specific fragments of MITL are decidable, while the full logic is undecidable.

Finally, Asarin et al. [2] propose symbolic techniques for controller synthesis in both discrete and timed systems, aiming to manage large state spaces without relying on exhaustive enumeration, by leveraging logical formulas. This line of work has been further developed through contributions involving the use of zones [12].

2 Preliminaries

We begin by defining the arenas on which our games are played. These arenas are essentially Alur-Dill timed automata [1], augmented with both discrete and continuous transitions, which are partitioned into transitions controlled by one player and transitions controlled by the other. We then define a family of functions that associate a game to each arena. These functions correspond to the semantics applied to arenas. A game, in this context, is a transition system equipped with a winning condition for one player; the objective of the other player is adversarial and defined as the complement of the first player's objective.

Clocks, Valuations, and Guards

An *interval* is a nonempty convex subset of the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. Intervals can be left-open or left-closed, right-open or right-closed, and bounded or unbounded. An interval takes one of the following forms: $[a, b]$, $[a, b)$, $[a, \infty)$, $(a, b]$, (a, b), (a, ∞), with $a, b \in \mathbb{N}$ and $a \leq b$. We write $\emptyset$ for any empty interval.

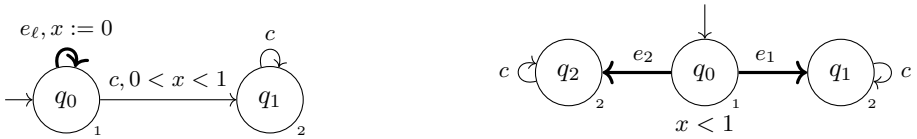
Let X be a finite set $\{x_1, x_2, \ldots, x_n\}$ of variables called clocks. An *atomic clock constraint* is a formula of the form $x \in I$ where I is an interval. A *guard* is a Boolean combination of atomic clock constraints. We denote by $\text{Guards}(X)$ the set of all guards on X. A *valuation* for the clocks in X is a function $v : X \to \mathbb{R}_{\geq 0}$. To simplify the notation, we sometimes write v_x for $v(x)$. We write $v \models g$ whenever the valuation v satisfies the guard g. $\text{Vals}(X)$ is the set of all valuations of X and $\mathbf{0}_X$ is the valuation that assigns 0 to all clocks of X. $\top \in \text{Guards}(X)$ is called *universal guard* and is satisfied by all valuations. $\bot \in \text{Guards}(X)$ is the *empty guard* and is never satisfied.

For $R \subseteq X$ (resp. $x \in X$), we write $v[R := 0]$ (resp. $v[x := 0]$) for the valuation that assigns 0 to all clocks $x \in R$ (resp. to x), and $v(x')$ to all the other clocks. For $t \in \mathbb{R}_{\geq 0}$, we write $v + t$ for the valuation that assigns the value $v(x) + t$ to each clock $x \in X$. Given two valuations v and v', we write $v \bowtie v'$, for $\bowtie \in \{<, \leq, =, \geq, >\}$ when $\forall x \in X : v(x) \bowtie v'(x)$. We can now define the syntax of timed games arenas.

Timed Game Arena

We study games played on arenas. An arena models the system under study. The definition of the game associated with an arena depends on the semantics under consideration. The game rules are determined by the specific semantics used.

A *timed game arena*, TGA for short, over actions $A = A_0 \uplus A_1$ and clocks X is a tuple $G = (Q, E, C, U_0, U_1, \alpha)$ where Q is a finite set of *locations*, $E \subseteq Q \times A \times \text{Guards}(X) \times 2^X \times Q$ is a finite set of labeled *edges*, $C : Q \to \text{Guards}(X)$ is an *invariant function*, $U_p : Q \to \text{Guards}(X)$ are the *no-wait functions* of the players $p \in \{0, 1\}$, and $\alpha : Q \to \{0, 1, \ldots, d\}$ is a *coloring function of the locations*. $\alpha(q)$ is the *color* of q (sometimes called its *priority*). The partition of A into A_0 and A_1 represents the possible actions of two players: Player 0, which is called the *Controller* and Player 1, the *Environment*. Given a TGA G, $\text{Guards}(G)$ is the set of guards that appear in G (in the image of C, U_0, or U_1, or as an edge guard in E). Figure 1a shows a TGA with one clock x and two locations. The number next to a location



(a) A TGA from [9]. Under the surprise semantics, the Controller can reach q_1 from q_0 by playing fast enough and retrying as many time as needed.

(b) A TGA where the Controller should win (by reaching either q_1 or q_2).

Figure 1 Two TGAs. Thick arrows are Environment's actions. The Controller aims to visit locations with even colors, indicated at their bottom right, infinitely often.

q is its color $\alpha(q)$. The Controller's actions set is $A_0 = \{c\}$ and the Environment's actions set (thick arrows) is $A_1 = \{e_\ell\}$. The guard below a location is its invariant. Intuitively, the invariant of a location q is a condition on the clocks that must be satisfied for the game to be in q. On this example, all the no-wait functions always return $\bot$ for both players and we do not display them. Intuitively, the no-wait functions of the players force them to play without a delay when the clocks have certain value. For example, if $U_0(q) = x < 1$, then Player 0 cannot let time elapse in location q whenever the clock x is smaller than 1. Those no-wait functions usually do not appear in the classical definitions of timed games, but they will turn out to be useful in the sequel.

To play games on such TGA, we need further assumptions. We extend all TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$ and clocks X with two extra actions $\bot_0$, $\bot_1$ and an extra clock z. The actions $\bot_p$ will be used in the game setting to mean that Player p will choose to stay in the current location: we assume that each location q bears two self-loops labeled respectively by $\bot_0$ and $\bot_1$. Formally, we let: $A_{\bot,p} := A_p \cup \{\bot_p\}$ for $p \in \{0,1\}$ and $E_\bot := E \cup \big\{(q, \bot_p, \top, \emptyset, q) \mid \forall q \in Q, p \in \{0,1\}\big\}$. When necessary, we call A_p the set of *regular actions* of p. The clock z will be used to measure elapsed time and is never reset or tested. We let $X_z = X \cup \{z\}$. From now on, we assume that all TGA are equipped with z, $\bot_0$ and $\bot_1$. Note that we do not formally fix an initial location in the definition of a TGA. We may give one on the figures to help intuition.

Alur-Dill semantics

All the semantics we will discuss in Section 3 are based on the following definitions. A *configuration* is a pair $(q, v) \in Q \times \mathrm{Vals}(X_z)$. The TGA can only be in such a configuration if v satisfies the invariant of q; and time can elapse, increasing the valuation, as long as the invariant stays satisfied. From a configuration (q, v), an edge (q, a, g, R, q') can be fired iff the current valuation v satisfies g. This leads to configuration (q', v'), where v' is obtained from v by resetting the clocks in R. Formally, given (q, v), we say that an action a is *enabled* whenever there is $(q, a, g, R, q') \in E$ s.t. $v \models g \wedge v[R := 0] \models C(q')$. We write $(q, v) \xrightarrow{\Delta, a} (q', v')$ with $\Delta \in \mathbb{R}_{\geq 0}$ to say that a is enabled in $v + \Delta$, leading from $(q, v + \Delta)$ to (q', v'), and that one can wait from (q, v) to $(q, v + \Delta)$ without violating the invariant $C(q)$ nor the no-wait function $U_p(q)$, with p s.t. $a \in A_{\bot,p}$. Formally, $(q, v) \xrightarrow{\Delta, a} (q', v')$ iff:

$$\begin{aligned}
&\forall \delta \in [0, \Delta] : v + \delta \models C(q) \wedge \exists (q, a, g, R, q') \in E_\bot : \\
&\big(v + \Delta \models g \wedge v' = v + \Delta[R := 0] \wedge v' \models C(q') \\
&\qquad\qquad \wedge a \in A_{\bot,p} \implies \forall \delta' \in (0, \Delta) : v + \delta' \not\models U_p(q)\big).
\end{aligned}$$

A *deadlock* occurs whenever there is a location-valuation pair (q, v) s.t. C limits the time one can stay in q starting from v and there is no action enabled in that time. Formally, there is a deadlock in (q, v) whenever $\nexists \Delta, a, q', v' : (q, v) \xrightarrow{\Delta, a} (q', v')$. From now on, we assume that our arenas are deadlock-free.

Clock Regions

We rely on the usual region construction [1] that we adapt slightly to account for the special clock z. It is based on a finite equivalence relation over clock valuations of X_z. A clock region is an equivalence class of this relation. A *region* is a pair (q, r) with r a clock region. Since there are finitely many regions, and regions form a time-abstract bisimulation, regions form a finite abstraction of the state space that is suitable for our purpose.

Given a TGA G, for each clock $x \in X$, let c_x be the largest constant in Guards(G) that involves x, where $c_x = 0$ if x does not occur in any guard, no-wait or invariant of G. Let v_1 and v_2 be two clock valuations in Vals(X_z). We let $X' = \{y \in X \mid v_1(y) < c_y\} \cup \{z\}$ denote the set of clocks y which have a value smaller than c_y in v_1 plus the special clock z. Then, v_1 and v_2 are *clock equivalent*, denoted $v_1 \equiv_G v_2$, iff:

1. for all $x \in X$: either $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$ or $\lfloor v_1(x) \rfloor > c_x$ and $\lfloor v_2(x) \rfloor > c_x$;
2. the ordering of the fractional parts of all clocks in X' is the same in v_1 and v_2;
3. for all clocks $x \in X'$: $v_1(x)$ is an integer if and only if $v_2(x)$ is an integer.

Note that the special case we apply to z can be interpreted as if z was automatically reset to 0 every time it reaches 1. Indeed, the first condition about the integer parts of the clocks does not constrain z, and the two other conditions constrain only the fractional part of z's value. Then, a *clock region* is an equivalence class of $\equiv_G$, and we write $[\![v]\!]_G$ for the clock region of the clock valuation v. It is well-known that $\equiv_G$ is a finite time-abstract bisimulation relation, hence that the number of clock regions is finite for all TGA [1].

Next, to obtain games, we will associate a *transition system* (whose states and transitions will define the possible game states and moves) and a *winning condition* to each TGA. We will consider *concurrent games*, where both players choose a move simultaneously and the game state is updated according to the choice of both players. In Section 3, we will discuss different possible ways to build a transition system for a given TGA.

Game

A *(labeled) transition system* over a potentially infinite set M of labels is a tuple $\Phi = (S, M, \rightarrow, \beta)$ where S is a potentially infinite set of states, $\rightarrow \subseteq S \times M \times S$ is a labeled transition relation, and $\beta : S \mapsto \mathbb{N}$ is a coloring function of the states whose image must be finite. As usual, we write $s \xrightarrow{m} s'$ instead of $(s, m, s') \in \rightarrow$.

To model concurrent games, the set of labels are of the form $M = M_0 \times M_1$. The elements of M_p are called the *moves* of Player p. Intuitively, the game is played *round by round* and each round is characterized by the current game state (assuming a fixed initial state $s_{in} \in S$). When the game is in state s, both players concurrently choose respective moves $m_0 \in M_0$ and $m_1 \in M_1$, and the game moves to any state s' s.t. $s \xrightarrow{m_0, m_1} s'$. Then, a new round starts and the game continues *ad infinitum*, forming an infinite *play*.

Formally, a *play from* s_{in} is an infinite sequence of the form $s^0 \left(m_0^i m_1^i s^i\right)_{i \in 1,2,3\ldots}$ s.t.: $s^0 = s_{in}$ and $s^{i-1} \xrightarrow{m_0^i, m_1^i} s^i$ for all $i \geq 1$. A *history* is a sequence as above that is finite. Let $K(\Phi)$ denote the set of all possible plays on a transition system Φ and $H(\Phi)$ be the set of its possible histories. A *run* is a history or a play.

We use *parity conditions* to define winning plays. Let $\rho = \left(s^0\left(m_0^i m_1^i s^i\right)_{i\in 1,2,3\dots}\right)$ be a play and β a coloring function of the states. We lift the coloring function β from states to runs: $\beta(\rho) = \left(\beta(s^i)\right)_{i\in 0,1,2\dots}$. We denote $\inf(\beta(\rho))$ the set of colors that occur infinitely often in ρ. Then, we write $\rho \models \text{Parity}_0(\beta)$ iff $\max(\inf(\beta(\rho)))$ is even and $\rho \models \text{Parity}_1(\beta)$ iff $\max(\inf(\beta(\rho)))$ is odd.

A *game* is obtained by associating a winning condition to a transition system. A game is thus a pair $\mathcal{G} = (\Phi, WC)$ where Φ is a transition system over a set $M_0 \times M_1$ and $WC \subseteq K(\Phi)$ a set of Player 0's *winning* plays. In games, players can play according to a strategy. A Player p's *strategy* is a function $\sigma_p : H(\Phi) \to M_p$ that associates a Player p's move to all histories. We denote by $Z_p(\Phi)$ the set of all strategies of Player p in Φ. A run $\rho = s^0(m_0^i m_1^i s^i)_{i\in 1,2,\dots}$ is played according strategies σ_0 and σ_1 if for every $i \geq 1$, $m_0^i = \sigma_0(h_i)$ and $m_1^i = \sigma_1(h_i)$ where h_i is the prefix of ρ up to s^{i-1}. Then, given two strategies σ_0 and σ_1, the *outcome* $\text{Out}^{s_{in}}(\sigma_0, \sigma_1)$ is the set of all plays from s_{in} played according to σ_0 and σ_1. A strategy $\sigma^\star$ of the Controller (Player 0) is *winning from* s_{in} iff $\text{Out}^{s_{in}}(\sigma^\star, \sigma_1) \subseteq WC$ **for all** Environment's strategy σ_1.

Finally, we assume that all games $\mathcal{G} = (\Phi, WC)$ are *well-defined*, i.e. (1) Φ is deadlock-free and (2) in all states, the moves available to a player do not depend on the move played by the other player. Formally, for any state $s \in S$, let $\text{Moves}_0(s) = \{m_0 \in M_0 \mid \exists m_1 \in M_1 : \exists s' \in S \text{ s.t. } s \xrightarrow{m_0,m_1} s'\}$. We define $\text{Moves}_1(s)$ symmetrically for Player 1. Then, for all $s \in S$, there are m_0, m_1, s' s.t. $s \xrightarrow{m_0,m_1} s'$, *and* for all $s \in S$, for all $m_0 \in \text{Moves}_0(s)$, for all m_1 in $\text{Moves}_1(s)$: there is s' s.t. $s \xrightarrow{m_0,m_1} s'$.

Deciding how to build a game from an arena is not a trivial problem, and we devote Section 3 to discuss several possibilities.

3 Three Semantics for Timed Games

In the present section, we discuss three possible ways to associate a game $\mathcal{G} = (\Phi, WC)$ to a TGA G, thereby providing three possible definitions of game semantics in a unified framework. This framework constitutes the first contribution of the paper. We first recall the "classical" semantics of timed games, and the "surprise" semantics of [9]. Then, as a second contribution we introduce a third, novel, semantics that refines and improves the "surprise" semantics. This refinement results in a semantics which is more natural, and that allows for efficient winning strategy computation. In Section 4, we will introduce a zone-based symbolic algorithm to compute winning strategies in this semantics.

Let us fix a TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$ and clocks X. We show how to build a game (Φ, WC). Since we consider timed games, the states of the transition system Φ are configurations (q, v) of the TGA G, and the moves of Player p are pairs $(\Delta, a) \in (\mathbb{R}_{\geq 0} \times A_{\perp,p})$. Intuitively, each round of the game starts in a state (q, v). Both players propose concurrently a delay Δ and an action a. The player with the shortest delay wins the round and his proposal (Δ, a) is played in G: we let Δ time units (t.u.) elapse, reaching $(q, v + \Delta)$; then follow an a-labeled edge. The special actions $\perp_0$ and $\perp_1$ mean that the player only wants to wait without following any edge.

Plays are now of the form $\rho = (q^0, v^0)(\Delta_0^1, a_0^1)(\Delta_1^1, a_1^1)(q^1, v^1)\dots$. We say that such a play is *time-divergent* iff the special clock z cannot be bounded along $v^0, v^1, v^2, \dots$, i.e. $\forall t \in \mathbb{R}_{\geq 0} : \exists i \in \mathbb{N} : v_z^i > t$. We write $\rho \models td$ if ρ is time-divergent.

We formally define a semantics $\mathcal{Q}$ as a function that associates a game $\mathcal{G} = (\Phi, WC)$ to a TGA G. We now present the three semantics mentioned above. Given a TGA G, $\mathcal{T}(G)$ is the timed game classically associated to G [1], $\mathcal{S}(G)$ is the game from [9] which accounts for the surprise effect, and $\mathcal{R}(G)$ is a new refinement of $\mathcal{S}(G)$. These three semantics are *sound*, i.e., given a deadlock-free arena, they give a game with a well-defined transition system.

Classical Timed Semantics $\mathcal{T}$

The "classical" timed semantics associated to G is the game $\mathcal{T}(G)$. In this settings, players can only play moves of the form $(0, a)$, meaning "play a immediately"; or $(\Delta, \bot)$, meaning "wait for Δ t.u.". For example, consider the TGA in Figure 1a. The objective of the Controller is to reach q_1 from q_0. To do so, he needs to wait in q_0 until x becomes non-null, because of the guard from q_0 to q_1. However, the Environment can always overtake him by playing $(0, e_\ell)$ that resets the clock. The Environment can keep playing that way, so, with the classical semantics, there is no winning strategy for the Controller in this game.

Formally, $\mathcal{T}(G) = (\Phi = (S,\ M,\ \to,\ \beta),\ WC)$ is defined as follows. The state set is $S = Q \times \mathrm{Vals}(X_z)$. The possible moves of Player p are $M_p = (\mathbb{R}_{\geq 0} \times \{\bot_p\}) \cup (0 \times A_p)$. The definition of the transition relation $\to$ follows the above intuition: $(q, v) \xrightarrow{(\Delta_0,a_0),(\Delta_1,a_1)} (q', v')$ iff there is $j \in 0, 1$ s.t. $\Delta_j \leq \Delta_{1-j}$ and $q' = q_j$, $v' = v_j$ with[1] $(q, v) \xrightarrow{\Delta_0,a_0} (q_0, v_0)$ and $(q, v) \xrightarrow{\Delta_1,a_1} (q_1, v_1)$. For the winning condition, we lift the coloring function α on the locations of G to the states of Φ, letting $\beta(q, v) = \alpha(q)$ for all states (q, v). Then, WC is defined as a parity condition: $WC = \{\rho \in K(\Phi) \mid \rho \models \mathrm{Parity}_0(\beta)\}$.

TEOS Surprise Semantics $\mathcal{S}$

The winning strategy of the Environment described above (always playing $(0, e_\ell)$ in q_0) is problematic since any play in its outcome is *not* time-divergent. This is regarded as a flaw in the model, since playing an infinite number of actions in a finite time is not realistic. This observation has prompted the introduction of the "surprise" semantics by de Alfaro et al. [9]. We called this semantics the "TEOS semantics", from the title of the paper, and denote it $\mathcal{S}$. It differs from $\mathcal{T}$ by the definition of the player moves and of the set of winning plays. In the TEOS semantics, we let $M_p = (\mathbb{R}_{\geq 0} \times A_{\bot,p})$. Thus, players can propose to wait *and, immediately after*, play an action. This is where the "surprise" stems from: it gives players the ability to *act quickly* and to take their opponent by surprise by playing an action after an arbitrarily short delay. However, plays where the time does not diverge should be avoided, so the game $\mathcal{S}(G)$ introduces a *blame function* bl to identify the player blocking the time. If the Controller is to blame in a play where time does not diverge, he looses the game even if the play satisfies his parity objective.

Formally, we let: $\mathrm{bl} : S \times M_0 \times M_1 \times S \to \{0, 1\}$ be s.t. $\mathrm{bl}(s, m_0, m_1, s') = 0$ iff $s \xrightarrow{m_0} s'$. Then, bl is lifted to runs by recording who is playing at each round: $\mathrm{bl}\left(s^0 \left(m_0^i m_1^i s^i\right)_{i\in 1,2,3\ldots}\right) = \left(\mathrm{bl}(s^{i-1}, m_0^i, m_1^i, s^i)\right)_{i\in 1,2,3\ldots}$. Player p will be to blame in play ρ iff p occurs infinitely often in $\mathrm{bl}(\rho)$. The Controller then wins iff it satisfies his parity objective on time-divergent plays *and* is not to blame if the time converges. Formally, for $p \in \{0, 1\}$, we write $\rho \models \mathrm{bl}_p$ iff $p \in \inf(\mathrm{bl}(\rho))$, and let: $WC = \{\rho \mid \left(\rho \models td \wedge \rho \models \mathrm{Parity}_0(\beta)\right) \vee \left(\rho \not\models td \wedge \rho \not\models \mathrm{bl}_0\right)\}$. The other components of $\mathcal{S}(G)$ are the same as for $\mathcal{T}(G)$.

Consider again Figure 1a, and the Environment's strategy $\sigma_1^\star$ that consists in always playing $(0, e_\ell)$ in q_0. In the TEOS semantics, the Controller has a strategy $\sigma_0^\star$ to win against $\sigma_1^\star$ that consists in always playing (ε, c), for $0 < \varepsilon < 1$. In the outcome of those strategies, the time converges, but the Controller is not to blame, so he wins those plays although his parity objective is not fulfilled. However, such a $\sigma_0^\star$ is *not* winning for the Controller against *all* strategies of the Environment, who can always play $(\frac{\varepsilon}{2}, e_\ell)$. Yet, the Controller now has a winning strategy $\sigma_{2^{-i}}$ that consists in playing $(2^{-i}, c)$ at the i-th round until q_1 is reached,

[1] Recall that $\rightharpoonup$ is defined above under the paragraph about the Alur-Dill semantics.

and then playing $(1, c)$ for ever. Against $\sigma_{2^{-i}}$, the only way for the Environment to avoid reaching q_1 is to always play smaller delays to win each round, such as $(2^{-i-1}, c)$ at the i-th round. Then, if q_1 is not reached, time converges and the Environment is to blame, so the Controller always wins, even if q_1 is never reached. This illustrates the *ability to act quickly* of the surprise semantics.

Refined Surprise Semantics $\mathcal{R}$

Our last semantics is a novel refinement of $\mathcal{S}$. The main idea with this new semantics is that we allow the Controller *not to act* in the states of the system where he has no (discrete) action at his disposal. Consider an example where the Controller needs to observe the system and act after the environment has chosen between two possible actions. Such a situation is modeled in Figure 1b, where the Controller observes the system in state q_0 (where he has no action) and waits for the Environment to play either e_1 or e_2, that will arrive within one time unit. In the $\mathcal{S}$ semantics, when in q_0, the Controller can only propose to play a sequence of the form $(\Delta_0, \bot), (\Delta_1, \bot), \ldots, (\Delta_i, \bot), \ldots$ where $\sum_{i\geq 0} \Delta_i < 1$, in order to satisfy the invariant. Then, the Environment can always play (t, e_1) with $t \geq t_i$ to let the Controller play at all rounds. In such plays, the time converges with the blame on the Controller who loses the game. Thus, surprisingly, the Environment wins by doing nothing, forcing the Controller to block time. Indeed, the TEOS semantics makes no distinction between a player who waits because he has no other choice and a player who voluntarily blocks the time. This prompts the first refinement of our semantics that consists in introducing a new move, denoted "$-$" and meaning that the player does not play anything.

Formally, we extend the set of possible moves of Player p to: $M_p = (\mathbb{R}_{\geq 0} \times A_{\bot,p}) \cup \{-\}$. We define an ancillary predicate $\mathrm{P}_p(q, v)$ which is true iff Player p can propose a regular action from (q, v), i.e. iff there are $\Delta \in \mathbb{R}_{\geq 0}$, $a \in A_p$, $q' \in Q$ and $v' \in \mathrm{Vals}(X_z)$ s.t. $(q, v) \xrightarrow{\Delta, a} (q', v')$. Observe that $\neg\, \mathrm{P}_p(q, v)$ implies $\mathrm{P}_{1-p}(q, v)$ because we have assumed that G is deadlock-free. Then, we adapt the transition relation as follows.

- $(q, v) \xrightarrow{(\Delta_0, a_0), (\Delta_1, a_1)} (q', v')$ iff both players can play: $\mathrm{P}_0(q, v) \wedge \mathrm{P}_1(q, v)$ and $\exists j \in 0, 1$ s.t. $\Delta_j \leq \Delta_{1-j}$ and $q' = q_j$, $v' = v_j$ with $(q, v) \xrightarrow{\Delta_0, a_0} (q_0, v_0)$ and $(q, v) \xrightarrow{\Delta_1, a_1} (q_1, v_1)$.
- $(q, v) \xrightarrow{(\Delta_0, a_0), -} (q', v')$ iff only Player 0 can play: $\neg\, \mathrm{P}_1(q, v)$ and $(q, v) \xrightarrow{\Delta_0, a_0} (q', v')$.
- $(q, v) \xrightarrow{-, (\Delta_1, a_1)} (q', v')$ iff only Player 1 can play: $\neg\, \mathrm{P}_0(q, v)$ and $(q, v) \xrightarrow{\Delta_1, a_1} (q', v')$.

With this change, the Controller no longer needs to propose delays from q_0 in Figure 1b. Instead, in q_0, the Controller does not play, and the Environment must play e_1 or e_2 to model an event; otherwise, the Environment loses by blocking time.

In the game $\mathcal{R}(G)$, we also adapt the *blame* predicate. In the $\mathcal{S}$ semantics, the blame only records which Player played last, no matter which delay he has played. We introduce a *lazy blame* policy, which differs from the classic blame in that a player who has played *a wait* $(\Delta, \bot_p)$ *long enough to leave the current clock region* is not blamed. While this change might sound inconsequential, it is important to obtain an efficient zone-based algorithm later in the paper. Formally, we let $\mathrm{lbl} : S \times M_0 \times M_1 \times S \to \{-, 0, 1\}$ be s.t. $\mathrm{lbl}((q, v), m_0, m_1, (q', v')) = -$ if $q' = q$ and $v \not\equiv_G v'$; *otherwise* $\mathrm{lbl}((q, v), m_0, m_1, (q', v')) = \mathrm{bl}((q, v), m_0, m_1, (q', v'))$. We lift lbl to runs like we did for bl and write $\rho \models \mathrm{lbl}_p$ iff $p \in \inf(\mathrm{lbl}(\rho))$. One can show that it is feasible to replace the blame $(-)$, assigned by lbl s.t. no one is blamed, with a repetition of the previous blame. This allows reducing the state space. In the next section, we focus on this $\mathcal{R}$ semantics and show how to solve the synthesis problem in this setting.

4 Solving the Synthesis Problem On $\mathcal{R}$

Given a well-defined game $\mathcal{G}$ and an initial state s_{in}, the *synthesis problem* asks to compute, if possible, a winning strategy from s_{in} for the Controller. Solving the synthesis problem on $\mathcal{R}$ is to solve the synthesis problem for all game $\mathcal{G} = \mathcal{R}(G)$ for all TGA G, and all initial configuration $(q, \mathbf{0}_X)$ for all location q of G.

In this section, we present an algorithm to solve the synthesis problem on $\mathcal{R}$. The main technicality to achieve this is to define an *augmented*[2] TGA G^+, and associate to it an appropriate semantics[3] to obtain a game $\mathcal{R}^+(G^+)$ on which the synthesis problem is easier to solve. This transformation moves just enough complexity from the $\mathcal{R}$ semantics to the TGA G^+: for instance, in G^+, the locations are augmented with information about the past. This allow us to obtain a game with a *pure parity objective*, i.e., a game whose objective is of the form $\{\rho \mid \rho \models \text{Parity}_0(\alpha)\}$ (while in $\mathcal{R}(G)$ the objective must also account for the blame).

Equiped with $\mathcal{R}^+(G^+)$, we could imagine a simple algorithm to solve the synthesis problem. It would work as follows. First, remove the concurrency of the game by letting the Controller make his choice first, then letting the Environment play against this move. This reduction is valid since we are trying to compute a winning strategy of the Controller against *any* strategy of the environment. One can also let the environment lift the non-determinism at this stage by letting him choose from whom the action will be played when the two have chosen the same delay. This is valid since we want a Controller's strategy s.t. *all the plays* in all the outcome sets are winning. Second, use the classical region construction to obtain a finite game. The resulting game is a finite, turn-based game played on graph with a parity objective, i.e. a *parity game*. Such games are well-studied [14] and enjoy positional strategies that can be computed by the classical Zielonka algorithm.

While this naive algorithm would be correct, we argue it would be inefficient and would not be benefit from the zone data-structure classically used to effectively handle set of regions. We argue in Section 4.4 that the fixed point computation would end up splitting many zones into individual regions, which would obliterate the advantage of using zones instead of regions. Instead, we propose a different algorithm (see Algorithm 1), which is inspired from the Zielonka algorithm, but works directly on the TGA G^+ (and not on the underlying TS) to avoid these pitfalls. In particular, we prove that, in G^+ (under semantics $\mathcal{R}^+$), one can solve synthesis by considering a restricted class of strategies only (Sections 4.2 and 4.4). We exploit these properties to obtain an efficient zone-based algorithm. Finally, from the winning strategies computed in $\mathcal{R}^+(G^+)$, we can extract winning strategies on $\mathcal{R}(G)$.

Throughout this section, we fix a TGA $G = (Q, E, C, U_0, U_1, \alpha)$ over actions $A = A_0 \uplus A_1$, with set of clocks X_z, and maximum color d. We also fix $\mathcal{R}(G) = \mathcal{G} = (\Phi, WC)$. We extend the notion of region to states (q, v). We say that two states $s = (q, v)$ and $s' = (q', v')$ are *region equivalent*, or *in the same region*, written $s \equiv s'$, whenever $q = q'$ and $v \equiv v'$. Then, a *region* is an equivalence class of states wrt $\equiv$. We write $[\![s]\!]_G$ for the region containing s. Those notions extend naturally to plays and histories of the same length.

Zones and zone federations

To manipulate convex sets of clock valuations, we rely on the classical zone data-structure. An *atom* is an inequality of the form $x \bowtie c$ or $x - y \bowtie c$, where $x, y \in X_z$ are clocks of G; $\bowtie \in \{<, \leq, =, \geq, >\}$; and c is an integer. Then, a *zone* is a finite conjunction of atoms. For

[2] As we will see later, "augmented" means that each location contains two extra discrete pieces of information: a color and the identifier of a Player.

[3] $\mathcal{R}^+$ is the same as $\mathcal{R}$ but adapted to the new syntax of G^+.

example, $(x \leq 5) \wedge (y - x \geq 3)$. Given a zone Z, its semantics $[\![Z]\!] \subseteq \text{Vals}(X_z)$ is the set of all valuations that satisfy all the atoms in Z, using the usual semantics for the comparison operators. To manipulate non-convex sets of valuations, we rely on *zone federations* [6], which are finite sets of zones. The semantics of a federation $F = \{Z_1, Z_2, \ldots, Z_n\}$ is the union $[\![F]\!] = [\![Z_1]\!] \cup [\![Z_2]\!] \cup \cdots \cup [\![Z_n]\!]$. Note that guards can be encoded as federations.

Given a federation F, one can compute efficiently [6] a federation $F_{\not\downarrow}$ representing all the valuations that can reach F by letting time elapse, i.e. $[\![F_{\not\downarrow}]\!] = \{v \mid \exists \Delta \in \mathbb{R}_{\geq 0} : v + \Delta \in [\![F]\!]\}$. Similarly, one can compute $Z[X := 0]$ s.t. $[\![Z[X := 0]]\!] = [\![Z]\!][X := 0]$ for a set of clocks X to reset. Given federations F_1 and F_2, one can also compute $F_1 \vee F_2$, $F_1 \wedge F_2$ and $F_1 - F_2$ s.t. $[\![F_1 \vee F_2]\!] = [\![F_1]\!] \cup [\![F_1]\!]$, $[\![F_1 \wedge F_2]\!] = [\![F_1]\!] \cap [\![F_1]\!]$ and $[\![F_1 - F_2]\!] = [\![F_1]\!] \setminus [\![F_2]\!]$. Federations also allow for inclusion and intersection checks. Thus, computing all the valuations of F that satisfy some guard or invariant g encoded into a federation F_g amounts to computing $F \wedge F_g$. We write $\top$ for the universal federation: $[\![\top]\!] = \text{Vals}(X_z)$; and $\bot$ for the empty: $[\![\bot]\!] = \emptyset$. For convenience, we identify a zone to the singleton federation containing it. Given a set of clocks X, $\mathbb{F}(X)$ is the set of all the zone federations one can create with the clocks of X. Federations naturally allow for the encoding of infinite set of states as finite set of pairs (location, federation): $[\![\{(q_1, F_1), (q_2, F_2), \ldots\}]\!] = \{(q, v) \in Q \times \text{Vals} \mid \exists i : q = q_i \wedge v \in [\![F_i]\!]\}$.

4.1 Reduction to a Parity Game

We apply ideas from [9] to reduce the game $\mathcal{R}(G) = (\Phi, WC)$ to a parity game $\mathcal{R}^+(G^+)$. First, we prove that for all strategies σ in $\mathcal{R}(G)$, there exists a strategy σ' that stops at each integer bound of z (called a *tick*) and is winning if σ is winning. A *z-stepped strategy* is a strategy that always plays a delay sufficiently small to prevent z from going beyond its next integer bound. Formally, a strategy σ is a z-stepped Player p's strategy when, for all histories h ending in a state (q, v), we have $\sigma(h) = (\Delta, a)$ with $v_z + \Delta \leq \lfloor v_z \rfloor + 1$. Given a TS Φ, $Z_p^z(\Phi)$ is the set of all z-stepped Player p strategies in Φ. Then:

▶ **Lemma 1** (z-stepped Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning z-stepped strategy σ'.*

Intuitively, σ' is built to mimic σ closely. For example, from a state where $z = 0.5$, σ plays $(1, a)$, then there is a winning strategy σ' that first plays $(0.5, \bot_p)$ to reach $z = 1$, then, if the proposed wait actually happened, plays $(0.5, a)$ to play a in the valuation chosen by σ.

Lemma 1 allows us to incorporate in the arena a modulo 1 behavior of z. Intuitively, in $G^+ = (Q^+, E^+, C^+, U_0^+, U_1^+, \alpha^+)$, z evolves in $[0, 1)$, and is always reset by the only possible action $\circlearrowleft$ whenever it reaches 1. On all locations q, the invariant becomes $C(q) \wedge z \leq 1$ and we add self-loops with action $\circlearrowleft$ that reset z when $z = 1$. We also check that $z < 1$ on all other edges.

Next, we need to encode the blaming mechanism in the colors of the parity game. Whenever there is a tick of z, we see color $2 + c$, where c is the largest color seen since the last tick (in the original game). On all the states where no tick has occurred, we see color $1 - b$, where b is the player to be blamed. So, if the Controller is to be blamed in a time-convergent play, color 1 will be seen infinitely often and the Controller will lose.

More formally, we have $Q^+ = \{(q, c, b) \in Q \times \{0, \ldots, d\} \times \{0, 1, -\} \mid c \geq \alpha(q)\}$. Each location G^+, is thus of the form (q, c, b), where c is the largest color seen since the last tick, and b is the blamed player. The set of actions is $A \cup \{\circlearrowleft\}$. No-wait and coloring functions are lifted to G's locations: for $f \in \{U_0, U_1, \alpha\}$, $f^+((q, c, b)) = f(q)$. Finally, the set of edges is $E^+ = \{((q,\ c,\ b),\ \circlearrowleft,\ z = 1,\ \{z\},\ (q,\ \alpha(q),\ b)) \mid (q,\ c,\ b) \in Q^+\} \cup \{((q, c, b), a, g \wedge z < 1, R, (q', \max(c, \alpha(q')), b')) \mid (q, a, g, R, q') \in E, b' \text{ s.t. } a \in A_{\bot, b'}\}$.

Now that we have defined the arena G^+, we need to associate to it a semantics $\mathcal{R}^+(G^+)$. To this end, we define $\mathcal{R}^+$ as $\mathcal{R}$ except that: (1) players cannot play $(\Delta, \circlearrowleft)$ with $\Delta > 0$ and (2) the coloring function is β^+ defined as $\beta^+((q,c,b),v) = 1-b$ if $v_z < 1$, and $\beta^+((q,c,b),v) = c+2$ if $v_z = 1$. Hence, $\mathcal{R}^+(G^+)$ is a game with a pure parity condition, where the colors have been adapted to let any player that blocks the time lose the game. Further, players cannot play $(\Delta, \circlearrowleft)$ with $\Delta > 0$, which preserves the winner of the game by Lemma 1. Since $\mathcal{R}^+(G^+)$ is built from $\mathcal{R}$, it also incorporates the *refined surprise rule* (that removes the need for a player without an enabled regular action to play), and the *lazy blame policy* (that does not assign blame to a player who has waited long enough to exit the current clock region).

4.2 Further restricting the strategies

Let us now show that the *lazy blame policy* and the *refined surprise rule* allow us to further restrict the family of strategies we can consider to solve synthesis on $\mathcal{R}$.

First, we fix some vocabulary. We say that, in a given state, a delay δ is *short* if we stay in the same region after waiting δ t.u.. Otherwise, a delay Δ is long (we use δ to emphasize that a delay is short). A short wait (long wait, short action, long action) is a move of the form $(\delta, \bot_p)$ (resp. $(\Delta, \bot_p)$, (δ, a), (Δ, a)). Finally, a wait or action is *immediate* when $\delta = 0$, and *delayed* when $\Delta > 0$. Next, tick is a function that associates to all runs $s^0 m_0^1 m_1^1 s^1 \dots$ a sequence of Booleans $(t^i)_{i>0}$ s.t. $t^i = \top$ iff s^i is a tick. Formally, $\mathrm{tick}((q,v),(q',v')) = \top$ iff $(\mathrm{frac}(v'_z) = 0) \wedge (v_z \neq v'_z)$; $\mathrm{tick}(s^0) = \top$; and $\mathrm{tick}(\dots s^{i-1}(m_0^i, m_1^i)s^i) = \mathrm{tick}(s^{i-1}, s^i)$ for all $i \geq 1$. Then, a strategy σ is a *region strategy* whenever, for all pairs of history (ρ_1, ρ_2) s.t. $\rho_1 \equiv \rho_2$ and $\mathrm{tick}(\rho_1) = \mathrm{tick}(\rho_2)$, we have $\mathrm{last}(\rho_1) \xrightarrow{\sigma(\rho_1)} s_1$ and $\mathrm{last}(\rho_2) \xrightarrow{\sigma(\rho_2)} s_2$ with $s_1 \equiv s_2$. We denote the set of region strategies for Player p as $Z_p^{\mathrm{r}}(G)$. Then:

▶ **Lemma 2** (Region Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning region z-stepped strategy σ'.*

The previous lemma implies that if a strategy is winning, then any strategy that differs only in the chosen delays, but not enough to change the regions or ticks, is also winning. Indeed, by Lemma 2, any short delayed move can be replaced by an immediate one:

▶ **Corollary 3.** *If there is a Player p's winning strategy σ, then there is a Player p's winning strategy σ' that never plays short delayed moves.*

Next, we show that *split strategies* are sufficient. Intuitively, split strategies always propose either an immediate action or a long wait. Formally, a Player p's strategy σ is a *split strategy* if for all history h we have $\sigma(h) = (\Delta, a)$ with $(a \neq \bot_p \implies \Delta = 0) \wedge (\Delta > 0 \implies a = \bot_p)$.

We claim that, under the $\mathcal{R}$ semantics, for every winning strategy, there exists a split winning strategy. This might sound surprising since the ability to play the combination of a delay and a move seems to be at the heart of the notion of "surprise" in timed games. Let us explain why, *under our new $\mathcal{R}$ semantics*, it is not the case. First observe that, thanks to Corollary 3, players do not need to play *short actions* with a non-null delay. Next, we need to explain why playing *long actions* are not necessary either thanks to our new definition of the blame. To illustrate this, we consider the TGA on Figure 1a. Recall that the controller has a winning strategy $\sigma_{2^{-i}}$ that consists in playing $(2^{-i}, c)$ at the i-th round until q_1 is reached. That is, the Environment may prevent the Controller to play c, but the Controller is able to retry as long as needed. Consider now the split strategy $\sigma'_{2^{-i}}$ that plays $(2^{-i}, \bot_0)$ while on (q_0, v) with $v_x = 0$, and $(0, c)$ whenever in (q_0, v) with $v_x > 0$. We argue that this strategy is losing with $\mathcal{S}$'s blame policy, but winning in $\mathcal{R}$. Under $\mathcal{S}$, the

Environment wins by playing $(1, \bot_1)$ in $(q_0, v_x = 0)$, and $(0, e_\ell)$ in $(q_0, v_x > 0)$. Indeed, consider the following time-converging play in the outcome of $\sigma_{2^{-i}}$ and $\sigma'_{2^{-i}}$: $(q_0, v_x = 0) \xrightarrow{(0.5,\bot_0),(1,\bot_1)} (q_0, v_x = 0.5) \xrightarrow{(0,c),(0,e_\ell)} (q_0, v_x = 0) \xrightarrow{(0.125,\bot_0),(1,\bot_1)} (q_0, v_x = 0.125) \cdots$, where the non-determinism is always resolved for the Environment. In this play, every other move played by the Controller is a long wait (because we leave the region where the clock x is equal to 0). Under $\mathcal{S}$'s blame policy, the Controller is blamed for the time-convergence and looses the game. Now, consider $\mathcal{R}$'s lazy blame policy which does not blame players that *play a long wait*. Under this semantics, $\sigma'_{2^{-i}}$ is now a winning *split-strategy*. This example shows why, in the $\mathcal{R}$ semantics, we can restrict ourselves to such strategies:

▶ **Lemma 4** (Split Strategies Are Sufficient). *Let G be a TGA. If a player has a winning strategy σ in $\mathcal{R}(G)$, then one can construct from σ a winning split region z-stepped strategy σ'.*

4.3 Synthesis Algorithm

We propose Algorithm 1 to solve the synthesis problem on all TGA, under semantics $\mathcal{R}$. Inspired by Zielonka algorithm, our algorithm is recursive, it computes the winning partition $W = (W_0, W_1)$ of the state space of $\mathcal{R}^+(G^+)$, and thanks to a trivial modification we obtain the winning Controller's strategy from each state of W_0. By construction, from all winning strategies in $\mathcal{R}^+(G^+)$ we can construct a winning strategy in $\mathcal{R}(G)$. We first describe the objects that the algorithm operates on. We then explain the algorithm step by step, providing intuition for the proof of correctness.

Algorithm 1 Synthesis Algorithm for $\mathcal{R}$ semantics, with S the set of states of $\mathcal{R}^+(G^+)$, $[\![k]\!]_{\beta^+} \subseteq S$ the set of states with β^+-color k, and $\textsc{Attractor}(G, j, [\![k]\!]_{\beta^+})$ the attractor for Player j to $[\![k]\!]_{\beta^+}$.

```
1:  function Solve(G)
2:      k ← max(β⁺(S))
3:      j ← k mod 2 ; i ← 1 − j
4:      T_j ← Attractor(G, j, [[k]]_β⁺)
5:      if T_j = S then
6:          W_i ← ∅, W_j ← S
7:          return W
8:      W' ← Solve(G \_j T_j)
9:      if W'_i = ∅ then
10:         W_i ← ∅, W_j ← S
11:         return W
12:     T_i ← Attractor(G, i, W'_i)
13:     W'' ← Solve(G \_i T_i)
14:     W_i ← S \ W''_j, W_j ← W''_j
15:     return W
```

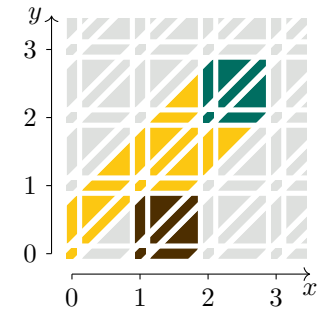


Figure 2 TPred Valuations.

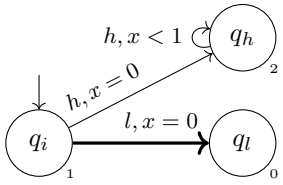


Figure 3 A TGA.

Overview

Algorithm 1 operates on the arena $G^+ = (Q^+, E^+, C^+, U_0^+, U_1^+, \alpha^+)$ and relies on the coloring function β^+, as defined in Section 4.1 (w.l.o.g., we assume that there is at least one state of each color from 0 to the maximum color d). It manipulates infinite sets of states of $\mathcal{R}^+(G^+)$ in the S, T_i and W_i variables. These set of states are symbolically represented as finite sets of pairs $(q, f) \in Q^+ \times \mathbb{F}(X)$, i.e. q is a location and f is a federation.

Like the Zielonka algorithm, Algorithm 1 solves a game by recursively solving subgames. These subgames are constructed so that the winning strategy for one of the player in the subgame remains a valid partial winning strategy in the current game.

The core components of Algorithm 1 are the ATTRACTOR function (lines 4 and 12), and a difference operator $\setminus_j$ (parameterized by a player number j) that reduces the size of the arena before recursive calls (lines 8 and 13). Intuitively, ATTRACTOR(G, j, T) returns all the states from which Player j can force game G to reach the target set of states T.

During each call, j is the player with the highest color in the game and i is his adversary. Hence, initially $j = 1$ iff d is odd and $i = 1 - j$. The attractor T_j is computed for j on the set of all states of maximal color (line 4). If all states of the arena are winning (line 5), the algorithm terminates with $T_j = S = W_j$. Otherwise, a subarena is computed by removing T_j from the initial arena G, and a recursive call is performed (line 8) to compute the winning states *for the other player*. If no states are discovered winning for him (line 9) the algorithm terminates with $T_j = S = W_j$. Otherwise, the attractor to these states T_i is removed and a recursive call is performed (line 13). Since Player i's attractor T_i was removed, we know that states winning for j in the subgame are exactly the states winning for j in the current game (line 14). To initialize this process, the algorithm first computes k, the largest β^+-color in G; j, the player for which k is a good color; and i, the other player. The target of the first attractor is $[\![k]\!]_{\beta^+}$, defined as the set of all states of maximal β^+ color.

In the end, given an initial location q_{in}, the Controller has a winning strategy $\mathcal{R}(G)$ from q_{in} whenever there is $((q_{in}, \alpha(q_{in}), -), F) \in W_0$ s.t. $\mathbf{0}_{X_z} \in [\![F]\!]$.

Attractor Computation

▶ **Definition 5** (Attractor). *The* attractor *for a Player p to a set of target states T is the set of all states from which p has a strategy that guarantees that the game evolves to a state of T (including T). Formally, given a set of states T, a Controller's attractor to T is the set of all states (q, v) s.t. $\exists \sigma_0 : \forall \sigma_1 : \forall \rho \in \mathrm{Out}^{(q,v)}(\sigma_0, \sigma_1)$: ρ visits T.*

An Environment's attractor to T is defined accordingly. It is the set of all states (q, v) s.t. $\forall \sigma_0 : \exists \sigma_1 : \exists \rho \in \mathrm{Out}^{(q,v)}(\sigma_0, \sigma_1)$: ρ visits T.

Given an arena G, a Player p and a set of target states T, the call to ATTRACTOR(G, p, T) returns the attractor for p to T. The efficient use of zones for the attractor computation relies on the properties of semantics $\mathcal{R}$ emphasized in 4.4. In order to compute attractors, we rely on the classical *one-step controllable predecessors*. The *one-step controllable predecessors* for a Player p to a set of target states T is the set of all states from which p will surely get into T in one step, whatever the move of the opponent, and is noted $\mathrm{CPre}_p(T)$. Assuming split strategies, the computation of $\mathrm{CPre}_p(T)$ itself can be broken down into several steps that we describe now.

TPred: Given a game $\mathcal{G}$ with states S, the *temporal predecessor* of a set of target states T avoiding a set of bad states B is, for $p \in \{0, 1\}$: $\mathrm{TPred}_p(T, B) = (T \setminus B) \cup \{s \in S \mid \exists \Delta \in \mathbb{R}_{\geq 0}$ s.t. $s \xrightarrow{(\Delta, \perp_p)} s', s' \in T \wedge (\nexists \Delta' \in \mathbb{R}_{\geq 0}$ s.t. $\Delta' \leq \Delta \wedge s \xrightarrow{(\Delta', \perp_{1-p})} s'' \wedge s'' \in B)\}$. For example, suppose we want to reach $\{(q, v) \mid v \models 2 \leq x < 3 \wedge 2 \leq y < 3\} = T$ (in green on Figure 2), but q has an Environment's outgoing edge with guard $(1 \leq x < 2 \wedge y < 1)$ (in dark) going into a location we want to avoid. So, B is $\{(q, v') \mid v' \models 1 \leq x < 2 \wedge y < 1\}$. Then $\mathrm{TPred}_0(T, B)$ is $\{(s, v'') \mid v'' \in f\}$ with f the set of valuations in yellow and green on the figure.

$\mathrm{TPred}_p(T, B)$ can be computed by manipulating federations of zones. Given a target set T, let $T_{\downarrow} = \{(q, f_{\downarrow}) \mid (q, f) \in T\}$. As shown in [6], $\mathrm{TPred}_p(T, B) = (T \setminus B) \cup (T_{\downarrow} \setminus B_{\downarrow})$ where $T \setminus B = \{(q, f) \in T \mid \nexists (q, f') \in B\} \cup \{(q, f \setminus f') \in T \in T \mid \exists (q, f') \in B\}$.

$\mathbf{DCPre_0}$: The Controller's discrete controllable predecessors of a set T is the set of states s s.t.: $\mathrm{DCPre}_0(T) = \{s \in S \mid (\exists a_0 \in A_0 : s \xrightarrow{0,a_0} s', s' \in T \wedge \nexists a_1 \in A_1 : s \xrightarrow{0,a_1} s'', s'' \notin T) \vee (\neg P_0(s) \wedge \exists a_1 \in A_1 : s \xrightarrow{0,a_1} s'', s'' \in T \wedge \nexists a_1' \in A_1 : s \xrightarrow{0,a_1'} s'', s'' \notin T)\}$. The second part of the disjunction captures the case where the Controller does not play in the $\mathcal{R}$ semantics.

$\mathbf{DCPre_1}$: The Environment's discrete controllable predecessor of a set of target states T is the set of states s s.t.: $\mathrm{DCPre}_1(T) = \{s \in S \mid \exists a_1 \in A_1 : s \xrightarrow{0,a_1} s', s' \in T \vee \forall a_0 \in A_0 : s \xrightarrow{0,a_0} s'', s'' \in T\}$.

$\mathbf{CPre_p}$: Finally, Player p's controllable predecessor of a set of target states T can be computed as: $\mathrm{CPre}_p(T) = \mathrm{TPred}_p(\mathrm{DCPre}_p(T), \mathrm{DCPre}_{1-p}(S \setminus T))$.

Now that we have the computation of $\mathrm{CPre}_p(T)$, we can use a standard fixed point algorithm to obtain the attractor:

▶ **Lemma 6.** *The attractor for a Player p to a set of states T is the closure* $(\mathrm{CPre}_p)^*(T)$.

Subgame Creation

Let us finally explain how we compute subgames with the $\setminus_i$ operator. Let us first explain why this requires careful consideration through the example in Figure 3. Here, the Controller wins by reaching either q_h or q_l. The Controller wins this game by playing $(0, h)$ in q_i: either this move gets played and q_h is reached, or the Environment plays $(0, l)$ and reaches q_l.

The algorithm starts by computing the attractor of the locations with maximal colors, i.e. $\textsc{Attractor}\big(G, 0, (q_h, x = 1)\big)$. This attractor is $(q_h, 0 \leq x \leq 1)$ and does not contain $(q_i, x = 0)$ since the Environment can force the transition from q_i to q_l in this state and thus avoid q_h. Next, the algorithm needs to solve a subgame from which we have removed at least the attractor $(q_h, 0 \leq x \leq 1)$. However, this example shows we need to remove more than simply location q_h and the transition from q_i to q_h. Indeed, in the same game without q_h, the Environment wins by waiting a non-null delay in q_i, which prevents reaching q_l (although this strategy does not make him win in the full game). This shows the difficulty: we want to remove transitions and states from the transition system while the algorithm works on the arena. Our difference operator is defined to take this into account.

Concretely, the $\setminus_i$ operator does not only remove locations and transitions, but also modifies the no-wait functions U_p to disable continuous transitions. Thus, let us first refine the TPred operator to account for the U functions: Every time we write $\mathrm{TPred}_p(T, B)$, we implicitly mean $\mathrm{TPred}_p(T, B \cup \overline{U})$ with $\overline{U} = \{(q, U_p(q)) \mid q \in Q)\} \setminus T$.

Given an attractor T_0 of Player 0, let T' denote the set of states s not in T_0 where there exists an enabled discrete transition of the Controller from s to some state in T_0. (So we know that there is an enabled Environment's discrete transition from s to a state out of T_0.) Then we define: $G \setminus_0 T_0 := (Q, E, C', U_0, U_1, \alpha)$ with $C'(q) = C(q) \setminus f$ if $\exists (q, f) \in T_0$ else $C(q)$, $U_1'(q) = U_1(q) \setminus f$ if $\exists (q, f) \in T'$ else $U_1(q)$. Removing a Player 1's attractor is straightforward. Given an arena $G = (Q, E, C, U_0, U_1, \alpha)$ and a set of states T_1 that is an attractor for Player 1, we define: $G \setminus_1 T_1 := (Q, E, C', U_0, U_1, \alpha)$ with $C'(q) = C(q) \setminus f$ if $\exists (q, f) \in T_1$ else $C(q)$.

We close this section by sketching the main ideas behind the proof of correctness of the algorithm.

▶ **Theorem 7.** *Algorithm 1 terminates and returns the winning sets of both players.*

Termination is established by showing that the depth of recursive calls is bounded, using the classical region construction. The set of regions $Q \times \text{CRegs}(G)$ of a game $\mathcal{R}(G)$ is finite. By definition, the attractor $\textsc{Attractor}(G, j, T)$ never returns an empty set, since it contains T. Thus, each recursive call is performed on a game with at least one region less than the game in the calling procedure, which establishes termination.

The correctness is established inductively. The **induction hypothesis** is that $\textsc{Solve}(G)$ returns a partition (W_0, W_1) of the states of $\mathcal{R}^+(G^+)$ s.t. p has a winning strategy from all states of W_p.

To prove the inductive step, we consider all the return statements. The partition returned on line 7 satisfies the induction hypothesis by properties of the attractor. On line 11, we have a Player j attractor T_j and a j-winning strategy from every state in the subgame $G \setminus_j T_j$. For the final return statement, note that T_i is an i-attractor to a set W_i' constructed s.t. i has a winning strategy from W_i' in G. Thus, T_i is winning for i. Finally, from the states of $G \setminus_i T_i$, if a player has a winning strategy in this subgame, then they also have a winning strategy in G. To prove that the return on line 15 is correct, we rely on the properties of our dedicated difference operator $\setminus_i$.

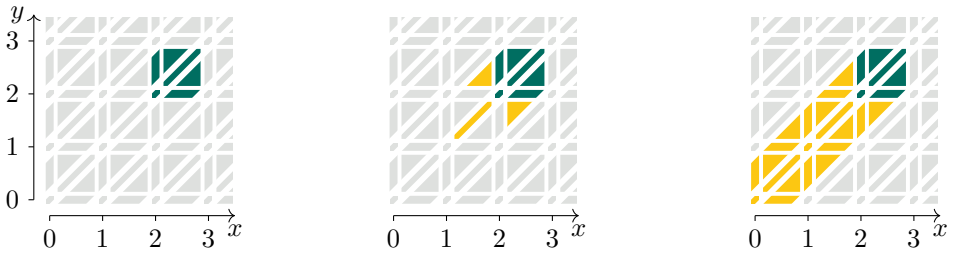
4.4 Efficient zone-based computation thanks to the $\mathcal{R}$ semantics

In this section, we explain why the definition of the $\mathcal{R}$ semantics is essential to obtain efficient zone-based algorithms. As mentioned before, the main takeaway message is that the $\mathcal{R}$ semantics (with the lazy blame policy) allows avoiding splitting zones into too many regions when computing the predecessor operator.

This problem is illustrated in Figure 4. First, remember that, in G^+, the states contain an extra bit indicating which player is to blame. This *discrete* information present in the states will be the source of the difficulties we are about to discuss. Assume we have game with a single location q, and assume we have obtained the symbolic state $(q, 1, T)$, where 1 indicates that the environment has the blame and T is the zone depicted in Figure 4a. First, observe that $\text{CPre}_0(\{(q, 1, T)\}) = \{(q, 1, T)\}$. Indeed, the Controller cannot force, in one step, to reach $(q, 1, T)$ from any other state, because the Controller has never the guarantee to play: either the Environment plays faster, or the non-determinism is resolved in favour of the Environment, who takes the blame (while letting the Controller play would have changed the blame bit to 0). Next, let us consider the computation of the Environment's Controllable Predecessors $\text{CPre}_1(\{(q, 1, T)\})$. In Figure 4b, we can observe that, if we had applied the "classical" blame policy to the $\mathcal{R}$ semantics, $\text{CPre}_1(\{(q, 1, T)\})$ would have a non-convex form, which means that it needs to be represented by a federation containing at least three zones. For example, region $x = 2 \wedge 1 < y < 2$ is not contained in $\text{CPre}_1(\{(q, 1, T)\})$ because the Controller could play a shorter delay than the one proposed by the Environment and take the blame, flipping the blame bit from 1 to 0. Using our lazy blame rule *that does not blame long waits* however preserves the convexity of $\text{CPre}_1(\{(q, 1, T)\})$, as seen in Figure 4c. The lazy blame avoid flipping the blame bit in the states and allows one to compute larger convex sets at once with the CPre operator.

5 Empirical Results

We have a proof of concept implementation which uses UppAal's federation library. We have tested it with artificial examples of varying sizes. We focus on time, as memory usage has remained relatively low in our experiments. These results aim to provide a general idea of the algorithm's behavior rather than measure its effectiveness precisely.



(a) A zone T. With the classic blame, we have $\{(q,1,T)\} = \mathrm{CPre}_0(\{(q,1,T)\})$.

(b) A federation $T'' \supset T$. With the classic blame, $\mathrm{CPre}_1(\{(q,1,T)\}) = \{(q,1,T'')\}$.

(c) A zone $T' \supset T$. With the lazy blame, $\mathrm{CPre}_0(\{(q,1,T)\}) = \mathrm{CPre}_1(\{(q,1,T)\}) = \{(q,1,T')\}$.

Figure 4 The lazy blame simplifies the controllable predecessor computation.

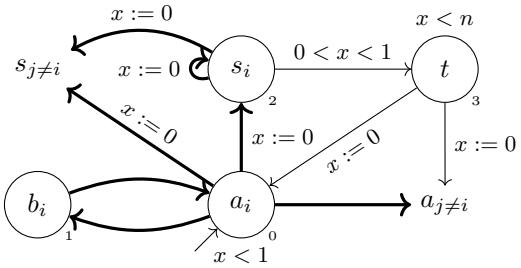


Figure 5 A TGA. Controller owns thin arrows.

Here are the main takeaways. As expected, the algorithm exhibits exponential time growth with respect to the number of clocks, polynomial time growth with respect to the number of distinct guards. We can observe that the algorithm effectively exploits the zone data-structure to capture the structure of the game. The problematic behavior explained in Figure 4b has not been observed. For the sake of brieveity, we now focus on a simple example.

Figure 5 shows a family of arenas. The arena is parametrized by an integer $n \geq 1$. It has $3n+1$ locations and $3n(n+1)$ edges, one clock x with $c_x = n$ and the largest color is 3. The locations are t and $\{a_i, b_i, s_i\}_{i\in\{1,\dots,n\}}$. In this game, the number of locations is growing linearly with n and the number of edges quadratically.

The calculation time is linear in the number of edges, showing that the zone data-structure effectively capture the structure of the game. We tested two variations: one where we change the invariant on t from $x < n$ to $x < 1$, and another where we replace the invariant $x < 1$ on a_i with $x < n$. The first variation (t with $x < n$) yields faster computation than the depicted case. The second variation yields slower computation. Both variations maintain linearity in the number of edges.

Finally, we observed that the tick interval impacts algorithm efficiency. This is easily understood: a tick interval that is too short results in smaller timed controllable predecessors. Experiments seem to indicate that using a tick interval equal to the largest constant found in guards and invariants yields good results. However, the question of how to choose an optimal tick interval, if possible, remains an open issue.

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

2 Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994*, volume 999 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1994. doi:10.1007/3-540-60472-3_1.

3 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 311–324. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.CONCUR.2015.311.

4 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005. doi:10.1007/11603009_5.

5 Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 825–837. Springer, 2007. doi:10.1007/978-3-540-73420-8_71.

6 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005. doi:10.1007/11539452_9.

7 Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire J. Tomlin and Mark R. Greenstreet, editors, *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002. doi:10.1007/3-540-45873-5_13.

8 Franck Cassez, Jan Jakob Jessen, Kim Guldstrand Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In Rupak Majumdar and Paulo Tabuada, editors, *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009. doi:10.1007/978-3-642-00602-9_7.

9 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003. doi:10.1007/978-3-540-45187-7_9.

10 Barbara Di Giampaolo, Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Safraless procedures for timed specifications. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010. doi:10.1007/978-3-642-15297-9_2.

11 Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2009. doi:10.1007/978-3-642-04368-0_12.

12 Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 3-5, 1997, San Francisco, CA, USA*, pages 14–24. IEEE Computer Society, 1997. doi:10.1109/REAL.1997.641265.

13 Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP semantics: from timed models to timed implementations. *Formal Aspects Comput.*, 17(3):319–341, 2005. doi:10.1007/S00165-005-0067-8.

14 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

On the Hardness of Approximating Distances of Quantum Codes

Elena Grigorescu
David R. Cheriton School of Computer Science, University of Waterloo, Canada

Vatsal Jha
Department of Computer Science, Purdue University, West Lafayette, IN, USA

Eric Samperton
Departments of Mathematics and Computer Science, Purdue Quantum Science and Engineering Institute, Purdue University, West Lafayette, IN, USA

Abstract

The problem of computing distances of error-correcting codes is fundamental in both the classical and quantum settings. While hardness for the classical version of these problems has been known for some time (in both the exact and approximate settings), it was only recently that Kapshikar and Kundu showed these problems are also hard in the quantum setting. As our first main result, we reprove this using arguably simpler arguments based on hypergraph product codes. In particular, we get a direct reduction to CSS codes, the most commonly used type of quantum code, from the minimum distance problem for classical linear codes.

Our second set of results considers the distance of a graph state, which is a key parameter for quantum codes obtained via the codeword stabilized formalism. We show that it is NP-hard to compute/approximate the distance of a graph state when the adjacency matrix of the graph is the input. In fact, we show this is true even if we only consider X-type errors of a graph state. Our techniques moreover imply an interesting classical consequence: the hardness of computing or approximating the distance of classical codes with rate equal to 1/2.

One of the main motivations of the present work is a question raised by Kapshikar and Kundu concerning the NP-hardness of approximation when there is an additive error proportional to a quantum code's length. We show that no such hardness can hold for hypergraph product codes. These observations suggest the possibility of a new kind of square root barrier.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases quantum codes, minimum distance problem, NP-hardness, graph state distance

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.34

Related Version *Full Version*: https://arxiv.org/abs/2509.21469

Funding *Elena Grigorescu*: Supported in part by NSF CCF-2228814 while at Purdue University.
Vatsal Jha: Supported in part by NSF CCF-2228814, NSF CCF-2330130, NSF CCF-2127806 and ONR Award N00014-24-1-2695.
Eric Samperton: Supported in part by NSF CCF-2330130.

Acknowledgements We thank Xuandi Ren for helpful correspondence, and some anonymous reviewers for valuable feedback.

1 Introduction

1.1 Quantum and classical distance problems

Scalable, fault-tolerant quantum computation is expected to require families of quantum error-correcting codes with larger and larger distance parameters, and CSS codes – introduced in [9] and [25] – provide a powerful and well-studied framework for building such families.


© Elena Grigorescu, Vatsal Jha, and Eric Samperton;
licensed under Creative Commons License CC-BY 4.0

In both the classical and quantum settings, it is a problem of fundamental importance to understand how best to compute distances of codes. Recently, Kapshikar and Kundu [19] showed that it is NP-hard to compute the distance of CSS codes, both in the exact sense and in various approximate senses (with various notions of polynomial-time reduction). Their proof relies heavily on the codeword-stabilized (CWS) framework, where a quantum code on n-qubits is defined using a graph G on n-vertices and a binary classical code C of length n. The goal of the present paper is to study the hardness of other related metrics for CWS codes, namely the minimum graph state distance associated with the graph G. To do this, we consider a problem in classical coding theory, which we refer to as the MinDistDualDist problem, and then prove its hardness. The latter problem seems to be of independent interest as it also provides the hardness of computing the distance of classical codes with rate equal to 1/2. Along with it, we provide simpler proofs of the results in [19] and try to shed light on the hardness of approximating distances with an additive error that is proportional to a code's length. In particular, the constructions used in the reduction of [19] hint at the possibility of a new kind of "square-root barrier" that we aim to better understand.

Before stating our results, let us first recall some of the basic notions of quantum and classical codes (if only to set notation), as well as formulate the precise distance problems that interest us. Despite being examples of quantum codes, CSS codes are defined using certain pairs of *classical* codes, so we review classical codes first. (Perhaps the most important insight of the early efforts on quantum error correction was that one can reduce the problem of their construction to certain slightly unusual constructions with classical codes.)

A *(classical, binary) code* C of *length* n is any subset $C \subseteq \mathbb{F}_2^n$, where $\mathbb{F}_2 = \{0, 1\}$ is the binary field and $\mathbb{F}_2^n$ the n-dimensional $\mathbb{F}_2$-vector space consisting of all row vectors of length n. If $C \leq \mathbb{F}_2^n$ is a linear subspace, then we call C a *linear* code. All classical codes considered in this paper will be linear, and so we will often not use the word "linear" when we should. The two most important parameters of a classical code are its *dimension* $k \stackrel{\text{def}}{=} \dim_{\mathbb{F}_2} C$ and its *distance*

$$d \stackrel{\text{def}}{=} \min\{d_H(x, y) \mid x, y \in C, x \neq y\} = \min\{d_H(x, 0) \mid x \in C, x \neq 0\},$$

where $d_H(x, y) \stackrel{\text{def}}{=} |\{i \in [n] \mid x_i \neq y_i\}|$ is the Hamming distance between vectors in $\mathbb{F}_2^n$. A classical linear code with such parameters is called an $[n, k, d]$ code.

There are two standard ways to present a classical linear code: with a *generator matrix* G, or with a *parity-check matrix* H. The former is any binary matrix whose row space equals C, while the latter is any matrix whose kernel equals C – or, more precisely (due to the difference between row and column vectors),

$$C = \{x \in \mathbb{F}_2^n \mid Hx^T = 0\}.$$

For a given parity-check matrix H, we let $C(H)$ denote the code corresponding to H. Parity-check matrices can be converted to generator matrices, and *vice versa*, in polynomial time. Moreover, when convenient, we may assume without loss of generality that the parity check matrix H of a $[n, k, d]$ code is a $(n - k) \times n$ matrix (in particular, has full rank $n - k$). A natural way of specifying a parity-check matrix H is via the systematic form where $H = [I_{n-k} : P_{(n-k)\times k}]$.[1] Given the parity-check matrix in systematic form, $G = [P^T_{k\times(n-k)} : I_k]$ will be a generator matrix for C.

The problem of calculating the distance of a classical code from its parity-check matrix has the following standard decision variant.

[1] For matrices A and B with equal number of rows, [A:B] refers to their augmented matrix

Classical Minimum Distance Decision Problem (MinDist)
Instance: A binary matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ and a non-negative integer t.
Output: YES if $d \leq t$. NO otherwise.

There are also important approximation variants of MinDist, either with a multiplicative error or an additive error. The decision variants of these are most conveniently formulated as promise problems with a gap. Let $\gamma \geq 1$ and $\tau > 0$.

Classical Minimum Distance Decision Problem with Multiplicative Gap γ ($\text{MultGapDist}_\gamma$)
Instance: A binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and a non-negative integer t.
Promise: Either $d \leq t$ or $d > \gamma t$.
Output: YES if $d \leq t$. NO if $d > \gamma t$.

Classical Minimum Distance Decision Problem with Additive Gap τn (AddGapDist_τ)
Instance: A binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and a non-negative integer t.
Promise: Either $d \leq t$ or $d > t + \tau n$.
Output: YES if $d \leq t$. NO if $d > t + \tau n$.

All three of these problems have been studied extensively, but the first two have received the most attention. The question of the complexity of MinDist was first raised in 1978 [2], and Vardy eventually showed that MinDist is NP-hard via deterministic Karp reduction [27]. Dumer, Miccancio, and Sudan later showed that $\text{MultGapDist}_\gamma$ and AddGapDist_τ are both NP-hard under RUR reduction for all $\gamma \geq 1$ and some $\tau > 0$ [14]. In the meantime, there have been various improvements [10, 1, 23, 4, 3], and it is now known that $\text{MultGapDist}_\gamma$ is NP-hard under deterministic Karp reduction. Remark 15 of [3] indicates that there is now a deterministic proof of the relatively near codeword problem in a specific parameter regime, but this does not appear to give the necessary result to derandomize the RUR reduction of [14] in the case of additive gaps (which goes through RNC). Nevertheless, Xuandi Ren kindly explained to us that the methods of [3] are in fact sufficient to show that AddGapDist_τ is NP-hard under deterministic Karp reduction for some $\tau > 0$ [24].

We need one more notion from classical codes before we can properly introduce CSS codes. Given a classical code C of length n, the *dual code* $C^\perp$ is another length n code defined as follows:

$$C^\perp \stackrel{\text{def}}{=} \{c' \in \mathbb{F}_2^n | \langle c', c\rangle = 0, \text{ for all } c \in C\}$$

where $\langle c', c\rangle$ is the usual $\mathbb{F}_2$ dot product. It is easy to see that a generator matrix for C is a parity-check matrix for $C^\perp$. In particular, if H is any parity-check matrix of a code C and G is a generator matrix for C, then $HG^T = 0$.

We are finally ready to briefly recall the description of quantum CSS codes.

To start, let us note that, in general, a *quantum error correcting code of length* n is any $\mathbb{C}$-linear subspace $\mathcal{C}$ of the Hilbert space of n qubits $(\mathbb{C}^2)^{\otimes n}$.[2] Similar to the classical setting, the *minimum distance* d_Q *of a quantum error-correcting code* is defined to be the minimum

[2] Thus, naively, every quantum error correcting code is "linear." However, the more important distinction in the quantum setting is between "additive" and "non-additive" codes. Additive quantum codes are understood as the proper quantum analog of linear classical codes, and non-additive quantum codes as the analogs of non-linear classical codes. We will not get into the details of this distinction here, except to note that all Pauli stabilizer codes are additive, and CSS codes are an important special case of Pauli stabilizer codes.

number of qubits where non-trivial errors must occur in order to effect a non-trivial logical error on the code-space. We call a quantum code of length n, with $\dim_{\mathbb{C}} \mathcal{C} = K$ and distance d a $((n, K, d))$ quantum code. If $K = 2^k$ happens to be a power of 2, then we call k the number of *logical qubits* in the code, and call the code a $[[n, k, d]]$ quantum code.

The seminal works [9] and [25] showed how to build certain quantum error-correcting codes – now called *CSS codes* – using any pair of classical binary linear codes (C_1, C_2) with $C_2^{\perp} \leq C_1$. If C_1 and C_2 have parameters $[n, k_1, d_1]$ and $[n, k_2, d_2]$, then the CSS code $CSS(C_1, C_2)$ has parameters $[[n, k_1 + k_2 - n, d_Q]]$ where

$$d_Q \stackrel{\text{def}}{=} \min\{wt_H(a) : a \in (C_1 \setminus C_2^{\perp}) \cup (C_2 \setminus C_1^{\perp})\}$$

is the (quantum) distance of $CSS(C_1, C_2)$. If the parity check matrices of C_1 and C_2 are H_1 and H_2, then the condition that $C_2^{\perp} \leq C_1$ is equivalent to to the condition $H_1 H_2^T = O$, and we will write $CSS(H_1, H_2)$ to mean $CSS(C_1, C_2)$, and may refer to H_1 and H_2 as the *quantum parity-check matrices* of the CSS code.

The quantum distance problems of primary interest in this work are the CSS analogs of the classical problems MinDist, MultGapDist and AddGapDist. Generalizing the first two is completely straightforward. As before, let $\gamma \geq 1$.

CSS Minimum Distance Decision Problem (CSSMinDist)
Instance: $H_X \in \mathbb{F}_2^{n-k_1 \times n}$ and $H_Z \in \mathbb{F}_2^{n-k_2 \times n}$ with $H_X H_Z^T = 0$, and a non-negative integer t.
Output: YES if $d_Q \leq t$. NO otherwise.

CSS Minimum Distance Decision Problem with Multiplicative Gap γ (MultGapCSSDist$_\gamma$)
Instance: $H_X \in \mathbb{F}_2^{n-k_1 \times n}$ and $H_Z \in \mathbb{F}_2^{n-k_2 \times n}$ with $H_X H_Z^T = 0$, and a non-negative integer t.
Promise: Either $d_Q \leq t$ or $d_Q > \gamma t$.
Output: YES if $d_Q \leq t$. NO if $d_Q > \gamma t$.

There is an important subtlety in generalizing AddGapDist to the CSS setting. We need *two* parameters for the problem: $\tau > 0$ as before and a new $\epsilon > 0$.

CSS Minimum Distance Decision Problem with Additive Gap τn^ϵ (AddGapCSSDist$_{\tau,\epsilon}$)
Instance: $H_X \in \mathbb{F}_2^{n-k_1 \times n}$ and $H_Z \in \mathbb{F}_2^{n-k_2 \times n}$ with $H_X H_Z^T = 0$, and a non-negative integer t.
Promise: Either $d_Q \leq t$ or $d_Q > t + \tau n^\epsilon$.
Output: YES if $d_Q \leq t$. NO if $d_Q > t + \tau n^\epsilon$.

As far as we are aware, the work of Kapshikar and Kundu [19] is the first to study these three problems explicitly, and they showed each is NP-hard. More precisely, they showed that CSSMinDist is NP-hard under Karp reduction, MultGapCSSDist$_\gamma$ is NP-hard under polynomial-time RUR reduction for every $\gamma \geq 1$, and there exists a $\tau > 0$ such that AddGapCSSDist$_{\tau,\epsilon}$ is NP-hard under polynomial time RUR reduction for every $\epsilon < \frac{1}{2}$. In fact, thanks to [10, 1, 3], their work is sufficient to establish that MultGapCSSDist$_\gamma$ is NP-hard under deterministic Karp reduction. Similarly, [3, 24] imply that AddGapCSSDist$_{\tau,\epsilon}$ (for some τ and all $\epsilon \leq \frac{1}{2}$) is hard under deterministic Karp reduction. We summarize these results:

▶ **Theorem 1** ([19]). *Each of the following is* NP*-hard under Karp reduction:*

- CSSMinDist
- MultGapCSSDist$_\gamma$ *for all* $\gamma \geq 1$
- AddGapCSSDist$_{\tau,\epsilon}$*, for each* $0 < \epsilon \leq \frac{1}{2}$ *and some* $\tau > 0$ *(depending on* ϵ*)*

Intriguingly, there are as yet no known hardness results for AddGapCSSDist$_{\tau,\epsilon}$ when $\frac{1}{2} < \epsilon \leq 1$. We initiated the present work as a step towards attempting to resolve this matter, and include extensive discussion in Section 5. The reader is particularly encouraged to consider Proposition 22 in Subsection 5.1 after reading the next subsection.

1.2 Our results

Our first main result, presented in Section 3, is a new proof of Theorem 1. Where Kapshikar and Kundu employ the codeword stabilized (CWS) formalism introduced in [12], we employ the hypergraph product (HGP) construction of Tillich and Zémor [26]. We quote directly from Kapshikar and Kundu in order to identify the basic difficulty that must be overcome in order to reduce the classical MinDist problem to the quantum CSSMinDist problem:

> Note that, due to the orthogonality condition on CSS codes, one can not use an arbitrary pair C_1, C_2. If we want to reduce the classical minimum distance problem, starting with C_1, we need to find a code C_2, such that, C_2 satisfies the orthogonality condition and has minimum distance not less than C_1. One way to get around this is to use self-dual (or weakly self-dual) classical codes in the CSS construction. But it is not clear whether the hardness result for classical codes still holds under the restriction that the code is self-dual (or weakly self-dual) [19].

We now compare and contrast the two constructions.

On one hand, the great insight of [19] is that one can go around the above obstacle by using the CWS formalism – rather than CSS codes – to build a (non-CSS) Pauli stabilizer code $CWS(C, G)$ on n qubits in a way that combines a classical linear code C with a graph G, with no restrictions on C and G other than that the number of vertices in G equal the length of C. With some additional deterministic polynomial time overhead, this Pauli stabilizer code can then be converted to a CSS code with the same parameters [5]. The delicate part when using this construction to reduce MinDist to CSSMinDist is then the choice of the graph G. For the CWS code based on C and G, the quantum distance d_Q is known to satisfy the inequality $d_Q \leq \min\{d, d_G\}$ where d is the classical distance of C and d_G is the *minimum graph state distance*

$$d_G \stackrel{\text{def}}{=} \min\{wt_H(x \vee z) : A_G x^T = z^T\}$$

(here A_G is the adjacency matrix of G). The reduction from MinDist to CSSMinDist in [19] ultimately succeeds by combining an input code C to MinDist with a graph G from a very special family of C_4-free graphs introduced in [15] that appear to be close to "optimal" [16]. With such a carefully chosen G, one gets that the quantum distance d_Q of $CWS(C, G)$ equals the classical distance d of C, which implies Theorem 1.

On the other hand, rather than combine a graph with a classical code, the *hypergraph product code construction* can combine *any* two classical linear codes (of *any* lengths) in a way that directly yields a CSS code [26]. Specifically, if H_1 and H_2 are the parity-check matrices for C_1 and C_2, then the *hypergraph product code* $HGP(H_1, H_2)$ is the CSS code with quantum parity check matrices H_1' and H_2' defined as follows:

$$H_1' \stackrel{\text{def}}{=} [H_1 \otimes I : I \otimes H_2^T], \qquad H_2' \stackrel{\text{def}}{=} [I \otimes H_2 : H_1^T \otimes I].$$

It is straightforward to verify that $H_1'(H_2')^T = 0$, and hence, this defines a valid CSS code. Moreover, if H_1 and H_2 have full rank and C_1 and C_2 have parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$, then [26] show that $HGP(H_1, H_2)$ is a code with parameters

$$[[n_1 n_2 + (n_1 - k_1)(n_2 - k_2), k_1 k_2, \min\{d_1, d_2\}]].$$

Put succinctly, when given full rank classical parity-check matrices, we have

$$HGP([n_1, k_1, d_1], [n_2, k_2, d_2]) = [[n_1 n_2 + (n_1 - k_1)(n_2 - k_2), k_1 k_2, \min\{d_1, d_2\}]] \quad (1)$$

Our proof of Theorem 1 is then hardly delicate: given a classical code C_1 with parity check matrix H_1, we choose H_2 simply to be the parity-check matrix of a repetition code of an appropriate length. The details are in Section 3. The basic idea is similar in spirit to the tensor-based techniques of [1] used to establish a deterministic reduction for the classical MultGapDist problem.

Before moving on to our other results, we note that the HGP construction is an important technique, as it was the first to break the square root barrier of distance at a constant rate. This construction was thus an important precursor to a long line of works that culminated in the proof of existence of "good" quantum LDPC codes [6], [20] and [13], and so it is desirable to have direct methods for exhibiting the intrinsic hardness of problems involving HGP codes.

Our second main set of results are related to graph state distance itself, which, as we already saw, is an important primitive in the CWS code construction. More generally, graph states have applications in quantum secret-sharing, quantum metrology and even quantum error-correction. They are extensively studied in [22], [18], [7], for example. While their precise definition is not so important for the present work, we note here that a graph state is a kind of Pauli stabilizer state, and thus can be understood as a quantum code that encodes 0 logical qubits (in other words, a single fault-tolerant quantum state). For interested readers, we include some details in Section 2.

A trivial upper bound on the minimum distance of the graph state corresponding to a graph $G = (V, E)$ is $\delta_G \stackrel{\text{def}}{=} \min_{v \in V} deg(v) + 1$. However, it can happen that $d_G \ll \delta_G$. Indeed, consider the graph $G = (V, E)$ one gets by removing a single edge – say, $\{v_1, v_n\}$ – from the complete graph K_n on n vertices. Then $\delta_G = n - 1$, but $d_G = 2$.

With these observations in hand, it is natural to consider the following distance problems for graph states.

Graph State Minimum Distance Decision Problem (GraphMinDist)
Instance: The adjacency matrix A_G of a simple graph and a non-negative integer t.
Output: YES if $d_G \leq t$. NO otherwise.

Graph State Minimum Distance Decision Problem with Multiplicative Gap γ ($\text{MultGapGraphDist}_\gamma$)
Instance: The adjacency matrix A_G of a simple graph and a non-negative integer t.
Promise: Either $d_G \leq t$ or $d_G > \gamma t$.
Output: YES if $d_G \leq t$ and NO if $d_G > \gamma t$.

Graph State Minimum Distance Decision Problem with Additive Gap τn^ϵ ($\text{AddGapGraphDist}_{\tau,\epsilon}$)
Instance: The adjacency matrix A_G of a simple graph and a non-negative integer t.
Promise: Either $d_G \leq t$ or $d_G > t + \tau n^\epsilon$.
Output: YES if $d_G \leq t$ and NO if $d_G > t + \tau n^\epsilon$.

▶ **Theorem 2.** *Each of the following is* NP*-hard under Karp reduction:*

- GraphMinDist
- MultGapGraphDist$_\gamma$ *for all* $\gamma \geq 1$
- AddGapGraphDist$_{\tau,\epsilon}$ *for each* $0 < \epsilon \leq \frac{1}{3}$ *and some* $\tau > 0$ *(depending on* ϵ*)*

Interestingly, even computing the promise version of GraphMinDist where $z = 0$ is NP-hard. More precisely, define the *graph state* X*-distance* $d_{X,G}$ by

$$d_{X,G} \stackrel{\text{def}}{=} \min\{wt_H(x) : x \in \mathbb{F}_2^n, A_G x^T = 0_{n\times 1}\}.$$

We can then define decision problems GraphMinDist^X, MultGapGraphDist$^X_\gamma$ and AddGapGraphDist$^X_{\tau,\epsilon}$ exactly as before, but with d_G replaced by $d_{X,G}$.

▶ **Theorem 3.** *Each of the following is* NP*-hard under Karp reduction:*

- GraphMinDist^X
- MultGapGraphDist$^X_\gamma$ *for all* $\gamma \geq 1$
- AddGapGraphDist$^X_{\tau,\epsilon}$ *for each* $0 < \epsilon \leq \frac{1}{3}$ *and some* $\tau > 0$ *(depending on* ϵ*)*

Unlike our proof of Theorem 1, our proofs of Theorems 2 and 3 do not use the hypergraph product construction. Instead, we use the tensor product of codes, together with the fact that simultaneously minimizing the distance of a (classical) code and its dual – a problem we call MinDistDualDist and define formally below – is hard. To prove this latter fact, we use an interesting recent construction of codes whose distance and dual distance are both "large" [21]. See Lemmas 17 and 18 for details. The hardness of MinDistDualDist furthermore implies another result in classical coding theory: the problem of bounding the distance of binary codes with rate equal to 1/2 is NP-Complete. To the best of our knowledge this does not appear in any earlier literature.

1.3 Organization

Section 2 briefly reviews the the Pauli stabilizer formalism and more carefully defines graph states. This section is not really needed for our proofs, but is included for interested readers. We prove Theorem 1 in Section 3. Theorems 2 and 3 are proved in Subsections 4.1 and 4.2, respectively. The final Section 5 includes extensive discussion, especially concerning the matter of hardness for AddGapCSSDist$_{\tau,\epsilon}$ when $\epsilon > \frac{1}{2}$.

2 Preliminaries

As we have already discussed the formalism for classical error-correcting codes in the introduction, we directly discuss the formalism for quantum error-correcting codes along with stabilizer formalism for quantum codes introduced in [17] and [8].

The state space of a single qubit is the two-dimensional Hilbert space, $\mathbb{C}^2$ and the n-qubit state space is the n-fold tensor product $(\mathbb{C}^2)^{\otimes n}$. As the state space of a single qubit is a 2-dimensional Hilbert space, hence any state $|\psi\rangle$ of a single qubit can be denoted by:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \text{ where } |\alpha|^2 + |\beta|^2 = 1 \text{ and } |0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}.$$

As the state space of n-qubits is the n-fold tensor product of $\mathbb{C}^2$ i.e. $(\mathbb{C}^2)^{\otimes n}$. Hence, a state $|\psi\rangle$ of n$-$*qubits* can be represented by:

$$|\psi\rangle = \sum_{x\in\mathbb{F}_2^n} \lambda_x\,|x\rangle\,,$$

such that $\sum_{x\in\mathbb{F}_2^n} |\lambda_x|^2 = 1$.

▶ **Definition 4.** *The Pauli Group on n-qubits is defined as:*

$$\mathcal{P}_n = \{i^\lambda M_1 \otimes M_2 \otimes ... \otimes M_n : \lambda \in \{0,1,2,3\},\ M_i \in \{I, X, Y, Z\} \text{ for all } i \in [n]\},$$

where $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ *and* $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

As $Y = iXZ$, hence, for every $g \in \mathcal{P}_n$ there exist $a, b \in \mathbb{F}_2^n$ such that $g = i^\lambda \bigotimes_{i=1}^n X^{a_i} Z^{b_i}$. A compact way to write this is $g = i^\lambda X(a)Z(b)$.

▶ **Definition 5.** *A stabilizer subgroup, $\mathcal{S}$, of $\mathcal{P}_n$ is an abelian subgroup not containing $-I$.*

▶ **Definition 6.** *For a stabilizer subgroup S of the Pauli group $\mathcal{P}_n$, the stabilizer code $C(\mathcal{S})$ associated with it is the joint $+1$-eigenspace of the operators in S:*

$$C(\mathcal{S}) = \{|\psi\rangle :\ g\,|\psi\rangle = |\psi\rangle \ \text{ for all } g \in S\}.$$

Moreover, a stabilizer code $\mathcal{S}$ is said to encode k-logical qubits if $dim(C(\mathcal{S})) = 2^k$

▶ **Fact 7.** *For a stabilizer group of order 2^k, the corresponding stabilizer code has dimension 2^{n-k}.*

▶ **Fact 8.** *For any operator $A \in M_{n\times n}(\mathbb{C})$ acting on $n-$qubits, we have:*

$$A = \sum_{a,b \in \mathbb{F}_2^n} \lambda_{a,b} X(a)Z(b),$$

where $\lambda_{a,b} \in \mathbb{C}$. Furthermore, the above representation of A as a sum of $X(a)Z(b)$ is unique.

▶ **Definition 9.** *The weight of an operator, $E \in M(\mathbb{C}_{n\times n})$, with representation:*

$$E = \sum_{a,b \in \mathbb{F}_2^n} \lambda_{a,b} X(a)Z(b),$$

is defined as:

$$wt(E) \stackrel{def}{=} \max\{wt_H(a \vee b) : \lambda_{a,b} \neq 0\}.$$

We can now define the distance of a quantum code.

▶ **Definition 10.** *For a quantum code on n qubits, it is said to have a distance d, if it can correct all errors E of weight $\leq \lfloor \frac{d-1}{2} \rfloor$.*

We now define the graph state, which we touched on in the introduction.

▶ **Definition 11.** *For a graph G on n-vertices, specified by an adjacency matrix A_G, the graph state is defined as the stabilizer code corresponding to the stabilizer group:*

$$\langle X(e_i)Z(u_i) : i \in [n]\rangle.$$

where e_i is the i^{th} standard basis vector and u_i is the i^{th} row of A_G.

We now define the graph state, which we touched on in the introduction.

▶ **Definition 12.** *For a graph G on n-vertices, specified by an adjacency matrix A_G, the graph state is defined as the stabilizer code corresponding to the stabilizer group:*

$$\langle X(e_i)Z(u_i) : i \in [n]\rangle.$$

where e_i is the i^{th} standard basis vector and u_i is the i^{th} row of A_G.

3 Proof of Theorem 1

Theorem 1 says that each of the three different problems CSSMinDist, MultGapCSSDist and AddGapCSSDist is NP-hard, and so we must provide three reductions. Each will start from the classical analogs MinDist, MultGapDist and AddGapDist, respectively.

To see that CSSMinDist is NP-hard, consider an input instance (H_1, t) of the MinDist problem, where H_1 is the parity-check matrix of a classical $[n, k, d]$ code C. We may assume H_1 is full rank (if it is not, then we may, in polynomial time, replace it with a new, smaller parity-check matrix that is). Now let H_2 be the (full rank) parity-check matrix for the classical repetition code of length n, which has parameters $[n, 1, n]$. We reduce to the hypergraph product code $HGP(H_1, H_2)$. By Equation 1, this hypergraph product code has parameters $[[n^2 + (n-k)(n-1), k, d]]$. This implies the hardness of CSSMinDist.

The reduction just shown builds a quantum code whose minimum distance equals the distance of the original classical code. Hence, even finding a constant multiplicative approximation to CSSMinDist is NP-hard. That is, MultGapCSSDist is NP-hard.

Finally, to prove the hardness of $\text{AddGapCSSDist}_{\tau,\epsilon}$, we reduce from AddGapDist_{τ}, which was shown to be NP-hard under RUR reduction for some τ by [14], although this argument can now be derandomized [3, 24].

To make things clearer, we define a variant of the classical AddGapDist_{τ} problem with an ϵ parameter.

Classical Minimum Distance Decision Problem with Additive Gap τn^{ϵ} ($\text{AddGapDist}_{\tau,\epsilon}$)
Instance : Binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and a positive integer t
Promise: Either $d \leq t$ or $d > t + \tau n^{\epsilon}$.
Output: YES if $d \leq t$ and NO if $d > t + \tau n^{\epsilon}$

▶ **Lemma 13.** *For every $\epsilon \in (0, 1]$, there exists $\tau \in (0, 1)$ for which $\text{AddGapDist}_{\tau,\epsilon}$ is NP-hard.*

Proof. To prove the lemma we will be using tensor codes, which are defined as follows:

▶ **Definition 14.** *The tensor product of two matrices*

$$A_{n\times m} := \begin{bmatrix} a_{11} & a_{12} & . & . & . & a_{1m} \\ a_{21} & a_{22} & . & . & . & a_{2m} \\ & & . & & & \\ & & . & & & \\ & & . & & & \\ a_{n1} & a_{n2} & . & . & . & a_{nm} \end{bmatrix} \quad \text{and} \quad B_{p\times q} := \begin{bmatrix} b_{11} & b_{12} & . & . & . & b_{1q} \\ b_{21} & b_{22} & . & . & . & b_{2q} \\ & & . & & & \\ & & . & & & \\ & & . & & & \\ b_{p1} & b_{p2} & . & . & . & b_{pq} \end{bmatrix}$$

is defined as:

$$A \otimes B := \begin{bmatrix} a_{11}B & a_{12}B & . & . & . & a_{1m}B \\ a_{21}B & a_{22}B & . & . & . & a_{2m}B \\ & & . & & & \\ & & . & & & \\ & & . & & & \\ a_{n1}B & a_{n2}B & . & . & . & a_{nm}B \end{bmatrix}$$

▶ **Definition 15.** *For classical codes C_1, C_2 with parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ and generator matrices G_1, G_2 respectively, the tensor product code, $C_1 \otimes C_2$, is defined as the code corresponding to the generator matrix $G_1 \otimes G_2$.*

We will use the following fact about tensor product codes:

▶ **Fact 16.** *The tensor product of classical codes, C_1 and C_2 with parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ (respectively) is a classical code with parameters $[n_1 n_2, k_1 k_2, d_1 d_2]$. Moreover, $(C_1 \otimes C_2)^{\perp} = C_1^{\perp} \otimes \mathbb{F}_2^{n_2} + \mathbb{F}_2^{n_1} \otimes C_2^{\perp}$ and has parameters $[n_1 n_2, n_1 n_2 - k_1 k_2, \min\{d_2, d_2\}]$ (See also Lemma 3.3 in [11]).*

We reduce from AddGapDist$_\tau$. Consider an instance (H, t) for the AddGapDist$_\tau$ problem, where τ is as defined in [14] (Theorem 32) such that AddGapDist$_\tau$ is NP-hard. Let C be the code of length n corresponding to the parity-check matrix H and let d be its minimum distance. Now, consider the tensor code $C^{'} = C \otimes \mathbb{F}_2^{\lceil n^{1/\epsilon - 1} \rceil}$. By Fact 16, it follows that the distance of $C^{'}$ is d and its length is $N = n \cdot \lceil n^{1/\epsilon - 1} \rceil$. We claim that $\min\{d, N^{\epsilon}\} = d$. Since $\min\{d, N^{\epsilon}\} \leq d$, it suffices to show that $\min\{d, N^{\epsilon}\} \geq d$, which follows from:

$$\begin{aligned} \min\{d, N^{\epsilon}\} &= \min\{d, (n \cdot \lceil n^{1/\epsilon - 1} \rceil)^{\epsilon}\} \\ &\geq \min\{d, (n \cdot n^{1/\epsilon - 1})^{\epsilon}\} \\ &= \min\{d, n\} \\ &= d. \end{aligned}$$

By definition $N = n \cdot \lceil n^{1/\epsilon - 1} \rceil$. Note that as $\lceil n^{1/\epsilon} \rceil \leq 2n^{1/\epsilon}$, we have $n^{1/\epsilon} \leq N \leq 2n^{1/\epsilon}$. If (H, t) is a YES instance for AddGapDist$_\tau$, then $d \leq t$, and as above it follows that $\min\{d, N^{\epsilon}\} \leq t$. If (H, t) is a NO instance for AddGapDist$_\tau$, then $\min\{d, N^{\epsilon}\} = d > t + \tau n \geq t + \tau^{'} N^{\epsilon}$, where $\tau' = \tau / 2^{\epsilon}$. This establishes the lemma. ◀

We now reduce the AddGapDist$_{\tau,\epsilon}$ problem to the AddGapCSSDist$_{\tau,\epsilon}$ problem, hence showing the hardness of the latter problem. For a given $\alpha \in (0, 1]$, consider the repetition code of length n^{α}. Let (H, t) be the input instance for AddGapDist$_{\tau,\epsilon}$ and let C be the code corresponding to parity-check matrix H having length n and distance d. The hypergraph product code obtained from the code C and the repetition code will be of length $n \cdot n^{\alpha} + (n - k) \cdot (n^{\alpha} - 1) = n^{'}$ and distance $d^{'} = \min\{d, n^{\alpha}\}$.

We claim that the above conversion of a classical code C to a hypergraph product code is a deterministic reduction from the AddGapDist$_{\tau,\epsilon}$ problem to the AddGapCSSDist$_{\tau,\epsilon}$ problem.

If (H, t) is a YES instance of AddGapDist$_{\tau,\epsilon}$, then $\min\{d, n^{\alpha}\} = d' \leq t$. This implies that the corresponding hypergraph product code has distance $d' \leq t$. If (H, t) is a NO instance, then $d' > t + \tau n^{\alpha} \geq t + \tau'(n')^{\frac{\alpha}{1+\alpha}}$, where $\tau' = \tau / 2^{\frac{\alpha}{1+\alpha}}$. This follows from the observation $n^{1+\alpha} \leq n' \leq 2n^{1+\alpha}$. Clearly, the gap between the YES instances and NO instances of AddGapCSSDist$_{\tau,\epsilon}$ is maximized for $\alpha = 1$, which is the case when the NO instance is $d' \geq t + \tau n^{1/2}$. ◀

4 Hardness for distances of graph states

4.1 Proof of Theorem 2

We now prove the hardness of computing the minimum graph state distance. We reduce MinDist to GraphDist via intermediate variants of MinDistDualDist. Similarly, to prove hardness for the gap version of GraphMinDist we will need the corresponding gap versions MultGapMinDistDualDist and AddGapDistDualDist as defined next.

Minimum Distance Dual Distance (MinDistDualDist)
Instance: Binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and positive integer t.
Output: If $\min\{d(C),\ d(C^\perp)\} \leq t$ then YES else NO.

Minimum Distance Dual Distance with Multiplicative Gap γ ($\text{MultGapMinDistDualDist}_\gamma$)
Instance: Binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and positive integer t.
Output: If $\min\{d(C),\ d(C^\perp)\} \leq t$ then YES and NO if $> \gamma t$.

Minimum Distance Dual Distance with Additive Gap τn (AddGapMinDistDualDist)
Instance: Binary matrix $H \in \mathbb{F}_2^{n-k\times n}$ and positive integer t.
Output: If $\min\{d(C),\ d(C^\perp)\} \leq t$ then YES and NO if $\min\{d(C),\ d(C^\perp)\} > t + \tau n^{1/3}$.

▶ **Lemma 17.** MinDistDualDist *is* NP*-hard. Furthermore, for every constant* $\gamma \geq 1$*, computing a* MultGapDistDualDist *approximation is* NP*-hard while there exists a* $\tau \in (0,1)$ *such that finding an additive approximation with error* $\tau n^{1/3}$ *is* NP*-hard.*

Proof. For a given parity-check matrix H, let C be the associated code to it with length n and minimum distance denoted by $d(C)$. Now, consider a code $C^{'}$, that comes from a family of codes $\{C_N^{'}\}$ where $C_N^{'}$ is a binary code of length N such that $\min\{d(C^{'}),\ d(C^{'})\} \geq \sqrt{N}$.

We consider the tensor code $\tilde{C} := C^\perp \otimes C^{'}$ and its dual $\tilde{C}^\perp := C \otimes \mathbb{F}_2^N + \mathbb{F}_2^n \otimes {C^{'}}^\perp$

One such family of codes was introduced in [21] (Theorem 2.9, Theorem 2.15).

▶ **Lemma 18** ([21])**.** *Let* $m \geq 5$ *be an odd integer with* $m \equiv 5 \mod 6$*. For every such* m*, there exists binary classical code* C_m *with parameters* $[2^m - 1, \frac{2^{m+1}-1}{3}, d \geq 2^{\frac{m-1}{2}} + 1]$*. Moreover,* $C_m^\perp$ *has parameters* $[2^m - 1, \frac{2^m-2}{3}, d^\perp \geq 2^{\frac{m-1}{2}} + 4]$*.*

While [21] does not provide any analysis of the time required to build their codes' parity-check matrices, one can check that this can be done efficiently. (To do so, one must use the fact that the finite field $\mathbb{F}_{2^m}$ of order 2^m with $m = O(\log n)$ can be constructed in time $O(poly(n))$.) We also use the observation that for every integer $n \geq 5$ there exists $m \equiv 5 \mod 6$ for which $2^m \leq n \leq 2^{m+6}$. The aforementioned observation along with the following fact allows us to increase the length of the code while preserving the distance.

▶ **Fact 19.** *For a linear code* C *of parameters* $[n,k,d]$ *and parity-check matrix* H*, the code corresponding to the parity-check matrix*

$$H^{'}_{(n-k)\times n'} = H \oplus I_{n'-n} = \begin{bmatrix} H_{(n-k)\times n} & O_{(n-k)\times(n'-n)} \\ O_{(n'-n)\times n} & I_{(n'-n)\times(n'-n)} \end{bmatrix},$$

has parameters $[n^{'}, k, d]$*.*

Using Fact 16 about the tensor-product of codes, we have:

$$\min\{d(\tilde{C}), d(\tilde{C}^\perp)\} = \min\{d(C^\perp)d(C'), d(C), d({C'}^\perp)\} \leq d(C).$$

Given that the length of code C is n, and $d(C')$, $d({C'}^\perp)$ are both greater than or equal to $\sqrt{N}$, we get

$$\min\{d(\tilde{C}), d(\tilde{C}^\perp)\} \geq \min\{d(C^\perp)\cdot n, d(C), n\} \geq d(C),$$

for $N \geq n^2$. Hence, $\min\{d(\tilde{C}), d(\tilde{C}^{\perp})\} = d(C)$. For the error in additive approximation, as we are embedding code of length n in a space of length n^3, hence we get the additive approximation term with cubic error. ◀

We now reduce this intermediate problem to GraphDist.

Proof of Theorem 2. For an input instance (H, t) of MinDistDualDist consider the systematic form representation of H i.e., $H = [I : P]$. Now, consider the following symmetric matrix A_P corresponding to H, i.e.:

$$A_P = \begin{bmatrix} O_{n-k\times n-k} & P \\ P^T & O_{k\times n-k} \end{bmatrix}.$$

It can be verified that as A_P is a symmetric matrix hence it can be treated as the adjacency matrix of a graph (infact a simple graph as the diagonal entries are all zero.)

As mentioned in the definition of minimum graph-state distance, the minimum graph state distance of A_P is obtained by finding $x := (a_1, ..., a_n | b_1, ..., b_n) \neq 0$ with minimum $wt_H(a \vee b)$ such that

$$[I : A_P]x^T = 0 \tag{2}$$

Now, the system of equations given in the above equation can be broken down into the following system of equations.

$$\begin{bmatrix} I : P \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-k} \\ b_{n-k+1} \\ \vdots \\ b_n \end{bmatrix} = O_{n-k\times 1} \tag{3}$$

$$\begin{bmatrix} P^T : I \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_{n-k} \\ a_{n-k+1} \\ \vdots \\ a_n \end{bmatrix} = O_{k\times 1} \tag{4}$$

Let S_1, S_2 be the set of variables that correspond to the above two system of equations respectively. As the sets S_1 and S_2 are disjoint, the minimum graph state distance for A_P, is obtained by assigning one set of variables the value 0, and for the remaining equation, we can find the non-zero solution with the least Hamming weight. This implies that the minimum graph-state distance corresponding to the graph whose adjacency matrix is A_P is:

$$d_G = \min\{d(C), d(C^{\perp})\},$$

and completes the proof of the hardness of GraphMinDist. Moreover, the above reduction also shows hardness for MultGapGraphDist and AddGapGraphDist. ◀

As a corollary to Lemma 17, we obtain that computing the distance of classical linear codes having rate 1/2 is NP-complete.

▶ **Corollary 20.** *It is* NP*-hard under Karp reductions to compute the distance of constant rate linear codes with rate* $\in (0, 1/2)$*. Furthermore, it is* NP*-hard under Karp reduction to compute both constant factor approximations and* $n^{1/3}$ *additive approximations to the distance of constant rate linear codes with rate* $\in (0, 1/2)$*.*

Proof. Consider a code C with parameters $[n, k, d]$ and parity-check matrix $[I_{n-k} : P_{n-k\times k}]$. Now consider the matrix:

$$\begin{bmatrix} I_k & 0_{k\times n-k} & : & 0_{n-k\times k} & P_{n-k\times k} \\ 0_{n-k\times k} & I_{n-k} & : & P^T_{k\times n-k} & 0_{k\times n-k} \end{bmatrix}.$$

The above matrix can be treated as a parity-check matrix corresponding to a rate 1/2 code. Computing the distance of the above code is computing a non-zero vector $(x, y) \in \mathbb{F}_2^n \oplus \mathbb{F}_2^n$ of least Hamming weight for which:

$$\begin{bmatrix} I_k & 0_{k\times n-k} & : & 0_{n-k\times k} & P_{n-k\times k} \\ 0_{n-k\times k} & I_{n-k} & : & P^T_{k\times n-k} & 0_{k\times n-k} \end{bmatrix} \begin{bmatrix} x^T \\ y^T \end{bmatrix} = 0_{n\times 1}.$$

As in the proof of Theorem 2, one sees the distance of this code is $\min\{d(C), d(C^\perp)\}$. To prove the hardness of codes with having rate $\in (0, 1/2)$, we take a code of rate equal to 1/2 and pad it with appropriate number of zeros. By Fact 19, this preserves the distance but decreases the rate. ◀

4.2 Proof of Theorem 3

Having showed that GraphMinDist is NP-hard, one might wonder whether it is at least possible to compute in polynomial time the minimum number of X-errors that can be detected by the graph state. We show that this problem and its gap versions are also NP-hard under Karp reduction.

Proof of Theorem 3. We reduce the problem of MinDist to X-GraphMinDist. In our reduction, we construct a self-orthogonal code C' from the given input code C while ensuring that $d(C')$ is proportional to $d(C)$. We explain this conversion of an arbitrary code C to a self-orthogonal C' in the following claim.

▷ Claim 21. Let C be a $[n, k, d]$ be a binary linear code. Consider the map $\pi : \mathbb{F}_2^n \to (\mathbb{F}_2^n)^2$ defined as:

$$\pi(a) = (a, a).$$

Then the image of C, defined as:

$$\pi(C) = \{\pi(c) : c \in C\},$$

is a self-orthogonal linear code with parameters $[2n, k, 2d]$.

Proof. We first prove that the map $\pi(C)$ is a linear code of length $2n$. For any two vectors $\pi(c), \pi(c') \in \pi(C)$ and $\lambda\ \mu \in \mathbb{F}_2$ we have that:

$$\lambda\pi(c) + \mu\pi(c') = \lambda(c, c) + \mu(c', c') = (\lambda c + \mu c', \lambda c + \mu c').$$

As C is a linear subspace hence $\lambda c + \mu c^{'} \in C$, hence $(\lambda c + \mu c^{'}, \lambda c + \mu c^{'}) \in \pi(C)$.

We now prove that $dim(\pi(C)) = dim(C) = k$. This follows from the observation that the map π is an injective map. Indeed, $\pi(c) = (0,0)$ implies $(c,c) = (0,0)$, *i.e.* $c = 0$.

The distance of the linear code $\pi(C)$ is calculated by observing that

$$\begin{aligned}\min_{\pi(c)\in\pi(C):\pi(c)\neq(0,0)} wt_H(\pi(c)) &= \min_{c\in C:c\neq 0} \pi(c) \\ &= \min_{c\in C:c\neq 0} 2wt_H(c) = 2\min_{c\in C:c\neq 0} wt_H(c) = 2d_H(C).\end{aligned}$$

Now, to prove that $\pi(C)$ is self-orthogonal we need to show that any pair of codewords $\pi(c),\ \pi(c^{'}) \in \pi(C)$ is orthogonal to each other.

Consider the inner-product:

$$\langle \pi(c), \pi(c^{'})\rangle = \langle (c,c), (c^{'}, c^{'})\rangle = \langle c, c'\rangle + \langle c, c'\rangle = 0,$$

where the last equality is due to the fact that we are in the field, $\mathbb{F}_2$ which has characteristic 2. This proves the self-orthogonality of $\pi(C)$. ◁

Now we describe our reduction from MinDist to X-MinDist.

Let (H,t) be the instance for MinDist and let C be the corresponding linear code with parameters $[n,k,d]$. By claim 21, $\pi(C) \in (\mathbb{F}_2^n)^2$ is a self-orthogonal linear code with parameters $[2n,k,2d]$. The systematic form of the parity-check matrix for $\pi(C)$:

$$[I_{2n-k\times 2n-k} : P_{2n-k\times k}].$$

along with its generator matrix:

$$[P^T_{k\times 2n-k} : I_k],$$

can be obtained in polynomial time from H.

Now, consider the matrix:

$$A_P = \begin{bmatrix} I_k & P_{k\times 2n-k} \\ P^T_{2n-k\times k} & I_{2n-k}\end{bmatrix}$$

The matrix A_P is symmetric and hence can be as a graph (though not a simple graph). Now, consider vectors $(x,z) \in \mathbb{F}_2^{2n} \oplus \mathbb{F}_2^{2n}$ with $z = 0$ such that

$$[I : A_P](z,x)^T = 0_{2n\times 1}$$

or equivalently,

$$A_P x^T = 0_{n\times 1}.$$

Solving the above equation is equivalent to solving the following system of equations:

$$[I_k : P_{2n-k\times k}]x^T = 0_{2n\times 1}$$

and,

$$[P^T_{2n-k\times k} : I_k]x^T = 0_{2n\times 1}$$

Thus, we are finding those vectors $x \in \mathbb{F}_2^{2n}$ for which both $Hx^T = 0_{2n-k\times 1}$ and $Gx^T = 0_{k\times 1}$. In other words, we are finding vectors $x \in \mathbb{F}_2^{2n}$ for which $x \in \pi(C) \cap \pi(C)^{\perp}$. Since $\pi(C)$ is self-orthogonal, it follows that $x \in \pi(C)$.

This implies that we want to find a non-zero vector $x \in \mathbb{F}_2^n$ with the least Hamming weight with $x \in \pi(C)$. This is precisely the minimum Hamming distance of the classical code $\pi(C)$ which is $2d_H(C)$. This proves the hardness of X-GraphMinDist as well as X-MultGapGraphDist$_\gamma$ under Karp reduction for $\gamma > 1$. Further, as the length of $\pi(C)$ is $2n$, X-AddGapGraphDist$_\tau$ is NP-hard for some $\tau \in (0, 1)$. ◀

5 Discussion and outlook

5.1 Controlling ϵ in AddGapHGPDist$_{\tau,\epsilon}$

Let AddGapHGPDist$_{\tau,\epsilon}$ denote the version of the problem AddGapCSSDist$_{\tau,\epsilon}$ where the input CSS code is a hypergraph product code.

▶ **Proposition 22.** *For any fixed $\epsilon \in (1/2, 1]$ and $\tau > 0$, there does not exist a Karp reduction from* MinDist *to* AddGapHGPDist$_{\tau,\epsilon}$ *unless* P = NP.

Proof. Suppose otherwise. Then there is a deterministic polynomial-time algorithm that takes an input instance (H, t) of MinDist (where H is a binary parity-check matrix and t is a non-negative integer) to an instance (H_1, H_2, t') of AddGapHGPDist in a way that preserves the respective decision problems' output. Say, H_1 and H_2 are full-rank and corresponding code parameters are $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$. Recall that the hypergraph product codes corresponding to parity-check matrices H_1 and H_2 with full rank and corresponding code parameters $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ have parameters $[[n_1n_2 + (n_1 - k_1)(n_2 - k_2), k_1k_2, \min\{d_1,\ d_2\}]]$. Due to the symmetry of the construction, we can assume that $d_1 = \min\{d_1,\ d_2\}$.

We reiterate a subtle point: n_1 and n_2 are functions of n and hence really should be expressed as $n_1(n)$ and $n_2(n)$, but we will suppress this dependency in order to avoid overloading our notation. Now consider the following cases:

1. For all (H_1, H_2, t'), $\min\{n_1, n_2\} = O(1)$: If we assume that $n_1 = \min\{n_1, n_2\} = O(1)$ then one can just perform an efficient brute-force search to decide if there is a non-zero vector x with the properties needed. While if $n_2 \leq n_1$, then using the fact that $\min\{d_1, d_2\} \leq \min\{n_1, n_2\} = O(1)$, one can check if there is a non-zero vector x with $|x| \leq n_2 = O(1)$ such that $H_1x^T = 0$. This would show that MinDist admits a poly-time algorithm, which contradicts $P \neq NP$.
2. $n_1, n_2 = \omega(1)$: Consider a NO instance (H, t) of MinDist. By the definition of Karp reduction, it follows $(H_1, H_2, t^{'})$ is a NO instance of AddGapHGPDist$_{\tau,\epsilon}$. under the assumption $d_1 = \min\{d_1\ d_2\}$,this gives the following set of inequalities:

   $$d_1 > t^{'} + \tau(n_1n_2 + (n_1 - k_1)(n_2 - k_2))^{\epsilon} \geq t' + \tau(n_1n_2)^{\epsilon}.$$

   Assume $n_1 \leq n_2$ and $d_1 = \min\{d_1,\ d_2\}$. The other case can be treated similarly. Then $d_1 \geq t' + \tau {n_1}^{2\epsilon}$ and so $d_1/n_1 \geq t'/n_1 + n_1^{2\epsilon-1}$. Since $d_1 \leq n_1$, one obtains a contradiction for large enough n_1. ◀

There are a number of very intriguing questions that Proposition 22 stimulates. Do there exist an $1/2 < \epsilon \leq 1$ and a $0 < \tau < 1$ for which AddGapHGPDist$_{\tau,\epsilon}$ is in coNP? Or P? Or maybe BQP? We discuss the most intriguing question in the next subsection.

5.2 A new square-root barrier?

On one hand, our technique for proving the hardness of AddGapCSSDist used hypergraph product codes, and resulted in an additive approximation with a square-root error term. Moreover, Proposition 22 shows that there is, in some sense, no way to improve on this square-root term using hypergraph product codes. On the other hand, the proof of [19] was based on rather different methods that exploited graphs with certain extremal properties, and yet arrived at the same kind of square-root error term. While there is no analog of Proposition 22 found in [19], it appears that the graphs they use in their reduction are essentially optimal [16]. It is interesting that the same square-root term in the approximation appears in both reductions, and one inevitably wonders if there is a new kind of square-root barrier at play.

5.3 Can we get a linear approximation for minimum graph state distance and linear codes with rate 1/2?

Recall that we were able to prove hardness for approximating the minimum graph state distance and the hardness of approximating the distance of linear codes with rate equal to 1/2 with a cube root additive term. Can we improve it? Following our proof strategy, a natural way to get past the cube root barrier and get an improvement, say a square root approximation, would be to find an infinite family of efficiently computable codes $\{C_m\}$ for which $\min\{d(C_m),\ d(C_m^\perp)\} = \Omega(n_m)$, where n_m is the length of C_m. Whether such an infinite family of codes exists is a possible open direction. The possibility of achieving the hardness results without the need of such a family of codes is also a direction to pursue.

5.4 Quantum "nearest codeword problem"

In the classical setting, the nearest codeword problem (NCP) has played a useful role in understanding the complexity of distances of codes. This is because NCP is essentially a non-homogeneous version of the minimum distance problem. So it is natural to wonder about quantum analogs of the question. One subtlety is that the codewords of a quantum code are, strictly speaking, *not* vectors over $\mathbb{F}_2$, but rather vectors in some subspace of the Hilbert space $(\mathbb{C}^2)^{\otimes n}$. (We have avoided discussing this point much in this paper, but see Section 2.) From this perspective, the quantum nearest codeword problem should be a question about computing the orthogonal projections of states onto the codespace. There are other, discrete variants one can imagine that happen inside the symplectic $\mathbb{F}_2$ vector space that contains the stabilizers of a code. These questions would not be about codewords *per se*, but about Pauli error operators, e.g. given a Pauli error, what is the closest logical error (either in Hamming weight or symplectic weight)?

References

1 Per Austrin and Subhash Khot. A simple deterministic reduction for the gap minimum distance of code problem. *IEEE Transactions on Information Theory*, 60(10):6636–6645, 2014. doi:10.1109/TIT.2014.2340869.

2 E. R. Berlekamp, R. J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

3 Vijay Bhattiprolu, Venkatesan Guruswami, Euiwoong Lee, and Xuandi Ren. Inapproximability of finding sparse vectors in codes, subspaces, and lattices. *arXiv:2410.02636. To appear* FOCS25, 2025.

4 Vijay Bhattiprolu and Euiwoong Lee. Inapproximability of sparsest vector in a real subspace. *arXiv preprint arXiv:2410.02636*, 2024. doi:10.48550/arXiv.2410.02636.

5 S. Bravyi, B. M. Terhal, and B. Leemhuis. Majorana fermion codes. *New Journal of Physics*, 12(8):083039, 2010.

6 Sergey Bravyi and Matthew B. Hastings. Homological product codes. *In Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, pages 273–282, 2014. doi:10.1145/2591796.2591870.

7 Adán Cabello, Lars Eirik Danielsen, Danielsen, Antonio J. López-Tarrida, and José R. Portillo. Optimal preparation of graph states. *Physical Review A*, 83(4), 2011.

8 A. Robert Calderbank, Eric M. Rains Rains, Peter M. Shor, and Neil JA Sloane. Quantum error correction via codes over gf (4). *IEEE Transactions on Information Theory*, 44(4):1369–1387, 1998. doi:10.1109/18.681315.

9 A. Robert Calderbank and Peter W. Shor. Hardness of approximating the minimum distance of a linear code. *Physical Review A*, 54(2), 1996.

10 Q. Cheng and D. Wan. A deterministic reduction for the gap minimum distance problem. *IEEE Trans. Inf. Theory*, 58(11):6935–6941, 2012. doi:10.1109/TIT.2012.2209198.

11 Andrew Cross, Zhiyang He, Anand Natarajan, Mario Szegedy, and Guanyu Zhu. Quantum locally testable code with constant soundness. *Quantum*, 8, 2024. doi:10.22331/Q-2024-10-18-1501.

12 Andrew Cross, Graeme Smith, John A. Smolin, and Bei Zeng. Codeword stabilized quantum codes. *IEEE International Symposium on Information Theory*, pages 364–368, 2008. doi:10.1109/ISIT.2008.4595009.

13 Nicolas Delfosse and Matthew B. Hastings. Union-find decoders for homological product codes. *Quantum*, 5:406, 2021. doi:10.22331/Q-2021-03-10-406.

14 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003. doi:10.1109/TIT.2002.806118.

15 A. R. P. Erdős and V. T. Sós. On a problem of graph theory. *Studia Scientiarum Mathematicarum Hungarica*, pages 215–235, 1966.

16 Zoltán Füredi and Miklós Simonovits. The history of degenerate (bipartite) extremal graph problems. In *Erdős centennial*, pages 169–264. Springer, 2013.

17 Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, California Institute of Technology, 1997.

18 Marc Heinn, Jens Eisert, and Hans J. Breigel. Multiparty entanglement in graph states. *Physical Review A*, 69(6), 2004. doi:10.1103/PhysRevA.69.062311.

19 Upendra Kapshikar and Srijita Kundu. On the hardness of the minimum distance problem of quantum codes. *IEEE Transactions on Information Theory*, 69(10):6293–6302, 2023. doi:10.1109/TIT.2023.3286870.

20 Tali Kaufman and Ran J. Tessler. New cosystolic expanders from tensors imply explicit quantum LDPC codes with $\Omega(\sqrt{n \log kn})$ distance. *In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1317–1329, 2021.

21 Hai Liu, Chengju Li, and Cunsheng Ding. Five infinite families of binary cyclic codes and their related codes with good parameters. *Finite Fields and Their Applications*, 91, 2023. doi:10.1016/J.FFA.2023.102270.

22 Damian Markham and Barry C. Sanders. Graph states for quantum secret sharing. *Physical Review A*, 78, 2008.

23 Daniele Micciancio. Locally dense codes. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 90–97. IEEE, 2014. doi:10.1109/CCC.2014.17.

24 Xuandi Ren. Private communication, 2025.

25 Andrew M. Steane. Error correcting codes in quantum theory. *Physical Review A*, 77(2), 1996.

26 J. P. Tillich and G. Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2013. doi:10.1109/TIT.2013.2292061.

27 Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997. doi:10.1109/18.641542.

Scalable Learning of One-Counter Automata via State-Merging Algorithms

Shibashis Guha
Tata Institute of Fundamental Research, Mumbai, India

Anirban Majumdar
Independent researcher, Kolkata, India

Prince Mathew
Indian Institute of Technology Goa, Ponda, India

A.V. Sreejith
Indian Institute of Technology Goa, Ponda, India

Abstract

We propose One-counter Positive Negative Inference (OPNI), a passive learning algorithm for deterministic real-time one-counter automata (DROCA). Inspired by the RPNI algorithm for regular languages, OPNI constructs a DROCA consistent with any given valid sample set.

We further present a semi-algorithm for active learning of DROCA using OPNI, and provide an implementation of the approach. Our experimental results demonstrate that this approach scales more effectively than existing state-of-the-art algorithms. We also evaluate the performance of the proposed approach for learning visibly one-counter automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases active learning, passive learning, one-counter automata, RPNI

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.35

Related Version *Extended Version*: https://arxiv.org/abs/2509.05762

Supplementary Material *Software*: https://doi.org/10.5281/zenodo.17310292 [9]
Software: https://doi.org/10.5281/zenodo.14604419 [8]

Funding Prince Mathew and A.V. Sreejith would like to thank CEFIPRA for the project IFC/6602-1/2022/77. A.V. Sreejith would also like to thank the DST Matrics grant for the project MTR/2022/000788.

1 Introduction

Automata learning and verification. Automata learning constitutes a correct-by-construction synthesis technique aimed at inferring formal models – such as finite-state machines or Mealy machines – from observed system behaviour. It ensures that the inferred model remains consistent with the observed data. Closely linked to formal verification, automata learning plays a key role in model inference, facilitating formal analysis and validation of system behaviour. It may also be viewed as an inductive synthesis framework tailored for the construction of finite-state programs.

Active and passive learning of automata. The L^* algorithm [3] of Angluin is a foundational algorithm in active automata learning. It learns a minimal deterministic finite automaton (DFA) that accepts an unknown regular language, using a query-based learning model. The learner interacts with a teacher (oracle) through two types of queries, called membership queries and equivalence queries. The L^* algorithm guarantees the learning of the minimal DFA in a number of steps polynomial in the size of the minimal DFA and the length of the longest counterexample.

© Shibashis Guha, Anirban Majumdar, Prince Mathew, and A.V. Sreejith;
licensed under Creative Commons License CC-BY 4.0

Passive learning of DFA, for example, Regular Positive and Negative Inference (RPNI) [13], on the other hand, involves inferring a DFA from a fixed, finite dataset consisting of positive and negative examples of strings. The output of the algorithm is a DFA that accepts all positive examples and rejects all negative ones. The constructed DFA may depend on the input positive and negative examples. For arbitrary inputs, the DFA that is consistent with the input data may not be minimal.

One-counter automata and visibly one-counter automata. Deterministic one-counter automata [14] are a subclass of deterministic pushdown automata that operate with a single integer counter, which can be incremented, decremented, or tested for zero during transitions. They serve as a simple model for programs with minimal memory – more powerful than finite automata, but less expressive than general deterministic pushdown automata. Deterministic real-time one-counter automata (DROCA) [4] form a subclass of deterministic one-counter automata that do not have ε-transitions.

Visibly pushdown automata (VPA) [1, 2] are a restricted subclass of pushdown automata where the type of stack operation (push, pop, or no operation) is determined by the input symbol. This restriction allows VPA to retain much of the expressive power of pushdown automata while ensuring desirable closure properties and algorithmic properties similar to finite automata. Unlike pushdown automata, deterministic visibly pushdown automata are equally expressive as nondeterministic ones. Deterministic visibly one-counter automata (VOCA) are both DROCA and deterministic VPA.

Contributions. We present two new methods for learning DROCA, addressing both passive and active learning settings:

- Our first contribution (Section 3) is a passive learning algorithm, called OPNI. Given a labelled set S of words – partitioned into accepting and rejecting examples – along with counter values for all prefixes, OPNI synthesises a DROCA that is consistent with both the counter information and the acceptance labels.
- Our second contribution (Section 4) is an active learning procedure, OCA-L*. In this setting, the learner interacts with a teacher who possesses a target DROCA. The learner may issue membership, equivalence, and counter-value queries to infer the target automaton. OCA-L* adapts the MinOCA framework [11] by replacing its SAT-based subroutine with our passive learner OPNI, leading to a simpler and significantly more scalable method.

We implemented OCA-L* in Python and evaluated it on randomly generated DROCA. Our experiments (Section 5) show that OCA-L* outperforms the state-of-the-art MinOCA algorithm in terms of scalability: while MinOCA fails on most DROCA with more than 10 states, OCA-L* successfully learns most DROCA having 18 or fewer states. Additionally, we evaluated OCA-L* for the class of VOCA. In this case, OCA-L* demonstrates remarkable scalability, learning most VOCA with up to 60 states and nearly half with 100 states. In contrast, MinOCA fails to learn most VOCA with more than 20 states.

However, unlike MinOCA, OCA-L* need not necessarily learn the minimal DROCA.

Related works. In the passive learning framework, the learner receives a set of labelled examples and synthesises a model consistent with them. The classical RPNI algorithm [13] learns a DFA in polynomial time from a set of accepting and rejecting words.

Active learning of finite automata via membership and equivalence queries was pioneered by Angluin's L^* algorithm [3]. In the context of learning one-counter automata, Fahmy and Roos [7] established the decidability of learning DROCA. Later, Neider and Löding [12] proposed an algorithm for learning VOCA, employing an additional partial-equivalence query.

Building on Neider and Löding's techniques, Bruyére et al. [5] proposed learning DROCA by combining counter-value queries – which return the counter value after processing a word. Mathew et al. [11] proposed an alternate algorithm (MinOCA) to learn DROCA. It employs SAT solvers to compute a minimal separating DFA. These algorithms have worst-case exponential runtime. Recent work [10] shows that active learning of DROCA is possible in polynomial time. However, the degree of the polynomial is large, making it impractical.

Organisation. The remainder of the paper is organised as follows. Section 2 presents the necessary preliminaries. In Section 3, we introduce our passive learning algorithm, OPNI. Section 4 describes the active learning procedure, OCA-L*. Section 5 reports on our experimental evaluation of OCA-L* and compares its performance with the state-of-the-art MinOCA algorithm. Finally, Section 6 concludes the paper.

2 Preliminaries

For a finite set S, we denote by $|S|$ the cardinality of S. Non-negative integers are denoted by $\mathbb{N}$, and $[i, j]$ denotes the interval $\{i, i+1, \ldots, j\} \subseteq \mathbb{N}$. The sign of a non-negative integer d (denoted by $sign(d)$) is 0 if $d = 0$ and 1 otherwise.

An alphabet denoted by Σ is a finite set of letters and Σ^* (resp., Σ^+) represents the set of all words including (resp., excluding) the empty word ε over the alphabet Σ. For a word $w = a_0a_1a_2 \ldots a_n$ and non-negative integers $i < j$, we use $w[i \cdots j]$ for the factor $a_ia_{i+1} \cdots a_j$ and $w[i]$ for the letter a_i. Given a set $S \subseteq \Sigma^*$ of words, we write $\mathsf{Pref}(S)$ to denote the set of prefixes of all words in S.

We now define the *length-lexicographic order* (denoted by $<_{\mathsf{llex}}$) on words. First, we fix an arbitrary total order $<_{\mathsf{lex}}$ on the letters in Σ. The order of words is inductively defined as follows: $u <_{\mathsf{llex}} v$ if either $|u| < |v|$, or $|u| = |v|$ and there exists $x, y, z \in \Sigma^*$ and $\sigma_1, \sigma_2 \in \Sigma$ such that $u = x\sigma_1 y$, $v = x\sigma_2 z$ and $\sigma_1 <_{\mathsf{lex}} \sigma_2$. Note that this ordering is total and well-founded. Note also that this order can naturally be extended to pairs of words as follows: $(u_1, v_1) <_{\mathsf{llex}} (u_2, v_2)$ if either $u_1 <_{\mathsf{llex}} u_2$, or $u_1 = u_2$ and $v_1 <_{\mathsf{llex}} v_2$.

2.1 One-Counter Automata

A *deterministic finite automaton* (DFA) is a tuple $D = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $\delta \colon Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting (final) states.

Given a DFA D, we sometimes write $q \xrightarrow{a} q'$ to denote $\delta(q, a) = q'$. The transition function δ extends naturally to words in Σ^* in the usual way: for $w = a_1a_2 \cdots a_n \in \Sigma^*$, we define $\delta(q, w) = \delta(\cdots \delta(\delta(q, a_1), a_2), \ldots, a_n)$. We will write $D(w)$ to denote $\delta(q_0, w)$. A word $w \in \Sigma^*$ is *accepted* by D if $D(w) \in F$. The language recognised by D, denoted as $\mathcal{L}(D)$, is the set of all accepted words. Two DFA D_1 and D_2 are equivalent if $\mathcal{L}(D_1) = \mathcal{L}(D_2)$.

▶ **Definition 1** (DROCA). *A* deterministic real-time one-counter automaton *(DROCA) is a tuple* $\mathcal{A} = (Q, \Sigma, q_0, \delta_0, \delta_1, F)$*, where* Q *is a finite nonempty set of states,* Σ *is the input alphabet,* $q_0 \in Q$ *is the initial state,* $\delta_0 : Q \times \Sigma \to Q \times \{0, +1\}$ *and* $\delta_1 : Q \times \Sigma \to Q \times \{0, +1, -1\}$ *are the transition functions, and* $F \subseteq Q$ *is the set of final states.*

Consider a DROCA $\mathcal{A}$. A configuration of $\mathcal{A}$ is a pair $(q, n) \in Q \times \mathbb{N}$, where q denotes the current state and n is the counter value. The configuration $(q_0, 0)$ is called the *initial configuration* of $\mathcal{A}$. For an $e \in \{-1, 0, +1\}$ and letter a, the *transition* between the configurations (p, n) and $(q, n + e)$ on the symbol a is defined if $\delta_{sign(n)}(p, a) = (q, e)$. We use $(p, n) \xrightarrow{a} (q, n + e)$ to denote this. The run of a word $w = a_1 \ldots a_n$ in I'm not not $\mathcal{A}$, if

it exists, is the sequence of configurations $(q_0, m_0) \xrightarrow{a_1} (q_1, m_1) \dots \xrightarrow{a_{n-1}} (q_n, m_n)$ where $m_0 = 0$. We will write $(q_0, m_0) \xrightarrow{w} (q_n, m_n)$ to denote such a sequence. Further, we say m_n is the *counter-effect* of w (denoted by $\mathsf{ce}_{\mathcal{A}}(w)$). Note that the counter values always stay non-negative, implying a decrement is not permitted from a configuration with a zero counter value. We will say that w is accepted by $\mathcal{A}$ if and only if $q_n \in F$.

The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all words accepted by $\mathcal{A}$. Similar to the case of DFA, we will write $\mathcal{A}(w)$ to denote the state in $\mathcal{A}$ that is reached after reading w. The definition of language equivalence is also similar to that above. Note that there are no ϵ-transitions in a DROCA. We say two DROCA $\mathcal{A}$ and $\mathcal{B}$ are *counter-synchronous* if $\mathsf{ce}_{\mathcal{A}}(w) = \mathsf{ce}_{\mathcal{B}}(w)$ for all words w.

A VOCA is a DROCA[1] where the input alphabet Σ is a union of three disjoint sets $\Sigma_{call}, \Sigma_{ret}$, and Σ_{int}. The VOCA increments (resp. decrements) its counter on reading a symbol from Σ_{call} (resp. Σ_{ret}). The counter value is unchanged on reading a symbol from Σ_{int}.

▶ **Definition 2** (VOCA). *A visibly one-counter automaton (VOCA) is a tuple* $\mathcal{A} = (Q, \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}, q_0, \delta_0, \delta_1, F)$*, where* Q *is a finite nonempty set of states,* $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ *is the input alphabet,* $q_0 \in Q$ *is the initial state,* $\delta_0 : Q \times \Sigma \to Q \times \{0, +1, -1\}$*,* $\delta_1 : Q \times \Sigma \to Q \times \{0, +1, -1\}$ *are the transition functions, and* $F \subseteq Q$ *is the set of final states.*

The notions of counter-effect, runs and transitions for VOCA remain the same as those of DROCA. From the definition, VOCA are deterministic. The counter-effect of a transition is solely based on Σ, making the starting state and counter value irrelevant. For $\sigma \in \Sigma$:

$$\mathsf{ce}_{\mathcal{A}}(\sigma) = \begin{cases} 1, & \text{if } \sigma \in \Sigma_{call} \\ -1, & \text{if } \sigma \in \Sigma_{ret}, \text{ and} \\ 0, & \text{if } \sigma \in \Sigma_{init} \end{cases}$$

If $w = \sigma_1 \sigma_2 \dots \sigma_n$ for some $n \in \mathbb{N}$ and $\sigma_1, \dots, \sigma_n \in \Sigma$, then $\mathsf{ce}_{\mathcal{A}}(w) = \mathsf{ce}_{\mathcal{A}}(\sigma_1) + \mathsf{ce}_{\mathcal{A}}(\sigma_2) + \dots + \mathsf{ce}_{\mathcal{A}}(\sigma_n)$. Since the counter value can never go below zero, there will be words that do not have a valid run in a VOCA. These words will be considered as rejected.

2.2 RPNI: a passive learning algorithm for DFA

Learning framework. A pair of sets of words $\mathsf{S} = \mathsf{S}_+ \cup \mathsf{S}_-$ over an alphabet Σ is called a *sample set*, and words in S_+ (resp., S_-) are called *positive* (resp., *negative*) samples. A sample set S is called *consistent* if it satisfies the condition: $\mathsf{S}_+ \cap \mathsf{S}_- = \emptyset$. Typically, in a *passive learning* framework for DFA, a *learner* is given an alphabet Σ, and a sample set S over Σ, that is consistent, and the objective is to construct an automaton D such that D accepts all the words in S_+ and rejects all the words in S_-. In that case, overloading the notation, we will say that D is *consistent* with the sample set S.

RPNI algorithm. Here we recall RPNI, a state-of-the-art passive learning algorithm for DFA, originally proposed in [13]. Intuitively, the algorithm starts with a prefix tree acceptor (PTA) constructed from the positive samples S_+, and then iteratively merges states of the PTA in a specific order (precisely, the length-lexicographic order), keeping the resulting DFA consistent with the samples set S. A pseudo-code of our version of RPNI is given in Algorithm 1. Below, we explain different steps of the algorithm.

[1] Syntactically, in a VOCA, the δ_0 transition function includes -1. However, this is not semantically allowed since the counter value can never go below zero.

Algorithm 1 RPNI: a passive learning algorithm for DFA.

Input: An alphabet Σ, and a *consistent* sample set $\mathsf{S} = \mathsf{S}_+ \cup \mathsf{S}_-$ over Σ.
Output: A DFA D consistent with the sample.

1 *Construct PTA*: Build a prefix tree acceptor PT from S_+.
2 *Pairing*: $P = \{(u, v) \mid v <_{\mathsf{llex}} u \text{ and } u, v \in \mathsf{Pref}(\mathsf{S}_+)\}$.
3 *Ordering* P: Sort the elements of P according to the increasing $<_{\mathsf{llex}}$ order.
4 Initialize $D \leftarrow \mathsf{PT}$.
5 **while** *P is not empty* **do**
6 Pop (u, v) the minimal element from P.
7 **if** *$D(u)$ and $D(v)$ can be merged* **then**
8 $D \leftarrow \text{Merge}(D; D(u), D(v))$.
9 **return** D.

We first construct a PTA (line 1) from the positive samples S_+, which can formally be defined as a DFA $\mathsf{PT} = (Q, \Sigma, \delta, q_\varepsilon, F)$, where $Q = \{q_u \mid u \in \mathsf{Pref}(\mathsf{S}_+)\}$ represents the set of states corresponding to all prefixes of words in S_+, δ is defined as follows: for all $u, u\sigma \in \mathsf{Pref}(\mathsf{S}_+)$ with $\sigma \in \Sigma$, we have $\delta(q_u, \sigma) = q_{u\sigma}$, the initial state q_ε is the state corresponding to the empty word ε, and $F = \{q_u \mid u \in \mathsf{S}_+\}$ is the set of all states that correspond to words in S_+. We call a prefix u the representative of the state q_u. Note that PT accepts exactly the words in S_+.

We then aim to merge states of the PTA in the $<_{\mathsf{llex}}$ ordering of their representatives, while preserving the consistency of the resulting DFA with the negative samples. To this end, we consider all pairs of prefixes (u, v) in $\mathsf{Pref}(\mathsf{S}_+)$ such that u is greater than v with respect to the $<_{\mathsf{llex}}$ ordering (line 2), and sort this set in the increasing $<_{\mathsf{llex}}$ order (line 3). We then initialise the main loop of RPNI with PT (line 4), and continue until P becomes empty (line 5). Let D be the DFA at the beginning of an iteration. First, pop (u, v) which is the minimal element from P (line 6). Then perform a merge of the states $D(u)$ and $D(v)$ of D, *i.e.*, check whether the merged DFA is consistent with respect to the sample set. If that is indeed (resp., not) the case, we accept (resp., discard) the merge, and update D accordingly (lines 7-8). Finally, we terminate when P becomes empty, and return the corresponding DFA.

The procedure *Merge* (line 8) takes as input a DFA D and two states $D(u)$ and $D(v)$, and returns a new DFA D' that is obtained from D by merging those two states. We give a high-level idea of the procedure. First, if $D(u) = D(v)$, then $D' = D$. Otherwise, let us denote $D(u)$ and $D(v)$ by q_u and q_v, respectively. Also, recall that since (u, v) is in the ordered set, $v <_{\mathsf{llex}} u$. Let q' be such that there exists a transition $q' \xrightarrow{a} q_u$ in D for some $a \in \Sigma$. Then construct D' as follows: delete the state q_u from D, redirect all incoming transitions of q_u to q_v, and all outgoing transitions from q_u will now be outgoing transitions of q_v. To make D' deterministic, we may need to recursively *fold* the subtree of q_u into q_v: if q_u and q_v have successors q'_u and q'_v on the same letter a, respectively, then merge q'_u and q'_v as well, and repeat this process until there is no common successor remaining.

The termination and correctness of Algorithm 1 were established in [13]. In Section 3, we will use RPNI as a black-box in our passive learning algorithm for DROCA.

3 OPNI: a passive learning algorithm for DROCA

In this section, we introduce OPNI, a passive learning algorithm for DROCA.

Learning framework. For passive learning of DROCA, we enhance the input of the learning algorithm with the counter values of prefixes of all words in the sample set. More formally, we assume that along with a consistent sample set $\mathsf{S} = \mathsf{S}_+ \cup \mathsf{S}_-$ over an alphabet Σ, the learning algorithm is also provided with the counter-effects for all words in $\mathsf{Pref}(\mathsf{S})$ defined by the function $\mathsf{ce} : \mathsf{Pref}(\mathsf{S}) \to \mathbb{N}$. The learner aims to construct a DROCA $\mathcal{A}$ that satisfies the following two properties: (1) it accepts all words in S_+ and rejects all words in S_-, and (2) for all $w \in \mathsf{Pref}(\mathsf{S})$, $\mathsf{ce}(w) = \mathsf{ce}_{\mathcal{A}}(w)$. In this case, we will say that $\mathcal{A}$ is *consistent* with the sample set S.

As in [11], we define the following two functions.

1. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$. Define $\mathsf{Act} : \Sigma^* \to \{0,1\} \times \{0, 1, -1, \mathtt{x}\}^k$ which, given a word w, returns a tuple consisting of the sign of the counter value reached after reading w along with the effect on the counter on reading each letter from Σ after w. Formally, for any word w, and $i \in [0, k]$,

$$\mathsf{Act}(w)[i] = \begin{cases} sign(\mathsf{ce}(w)) & \text{if } i = 0 \\ \mathsf{ce}(w\sigma_i) - \mathsf{ce}(w) & \text{if } i > 0 \text{ and } \mathsf{ce}(w\sigma_i) \text{ is known} \\ \mathtt{x} & \text{if } i > 0 \text{ and } \mathsf{ce}(w\sigma_i) \text{ is not known} \end{cases}$$

   Given words w_1, w_2, we say that $\mathsf{Act}(w_1)$ is similar to $\mathsf{Act}(w_2)$ (denoted $\mathsf{Act}(w_1) \sim \mathsf{Act}(w_2)$) if one of the following happens: (a) for all $i \in [1, k]$, we have $\mathsf{Act}(w_1)[i] = \mathtt{x}$ or $\mathsf{Act}(w_2)[i] = \mathtt{x}$ or $\mathsf{Act}(w_1)[i] = \mathsf{Act}(w_2)[i]$, or (b) the sign of the counter values for words w_1 and w_2 are different, that is, $\mathsf{Act}(w_1)[0] \neq \mathsf{Act}(w_2)[0]$. (As it will be clear later, when $\mathsf{Act}(w_1) \sim \mathsf{Act}(w_2)$, we might be able to merge the states reached after reading w_1 and w_2 respectively. The merging may be possible if the signs of the counter values are different, regardless of the other components in Act.) Otherwise, $\mathsf{Act}(w_1)$ and $\mathsf{Act}(w_2)$ are not similar, and we write $\mathsf{Act}(w_1) \not\sim \mathsf{Act}(w_2)$. For $\sigma \in \Sigma$, we write $\mathsf{Act}(w)|_\sigma$ to denote the entry in $\mathsf{Act}(w)$ that corresponds to the letter σ.
2. Define a function that, intuitively, given a word w, annotates each letter $w[i]$ of w with the sign of the counter-value reached, upon reading the prefix $w[0 \cdots i-1]$. To that end, for an alphabet Σ, we define the modified alphabet $\widetilde{\Sigma} = \bigcup_{\sigma \in \Sigma} \{\sigma^0, \sigma^1\}$. Then, given a word $w \in \Sigma^+$ and the counter-values ce of all its prefixes, we define the encryption function as follows: for $i \in [0, |w| - 1]$,

$$\mathsf{Enc}(w)[i] = \begin{cases} w[i]^0 & \text{if } i = 0 \\ w[i]^{sign(\mathsf{ce}(w[0 \cdots i-1]))} & \text{if } i > 0 \end{cases}$$

   Additionally, $\mathsf{Enc}(\epsilon) = \epsilon$.

▶ **Example 3.** Let $\Sigma = \{a, b\}$ and suppose the following counter-value information are given: $\mathsf{ce} = \{\varepsilon \to 0, a \to 1, b \to 0, ab \to 0, bb \to 1\}$. Then $\mathsf{Act}(a) = (sign(\mathsf{ce}(a)), \mathsf{ce}(aa) - \mathsf{ce}(a), \mathsf{ce}(ab) - \mathsf{ce}(a)) = (1, \mathtt{x}, -1)$. On the other hand, the effect of the Enc function on the word ab, for example, will be $a^0 b^1$.

Algorithm 2 OPNI: a passive learning algorithm for DROCA.

Input: An alphabet Σ, a *consistent* sample set $\mathsf{S} = \mathsf{S}_+ \cup \mathsf{S}_-$ over Σ, the function $\mathsf{ce} : \mathsf{Pref}(\mathsf{S}) \to \mathbb{N}$.
Output: A DROCA $\mathcal{A}$ consistent with the sample.

1 *Incorporate counter-actions*: Consider the modified sample set $\widehat{\mathsf{S}} = \widehat{\mathsf{S}}_+ \cup \widehat{\mathsf{S}}_-$ by replacing every word u in S with $\mathsf{Enc}(u)$.
2 *Enriching the alphabet*: Let $\Sigma_{\mathsf{Act}} = \{\mathsf{Act}(w) \mid w \in \mathsf{Pref}(\mathsf{S})\}$. Then, enrich the alphabet Σ with Σ_{Act} as follows: $\widehat{\Sigma} = \widetilde{\Sigma} \cup \Sigma_{\mathsf{Act}}$.
3 *Enriching the sample set*: For every word $w \in \mathsf{Pref}(\mathsf{S})$: add $\mathsf{Enc}(w) \cdot \mathsf{Act}(w)$ to $\widehat{\mathsf{S}}_+$, and for every other $\mathsf{op} \in \Sigma_{\mathsf{Act}}$ such that $\mathsf{op} \not\sim \mathsf{Act}(w)$, add $\mathsf{Enc}(w) \cdot \mathsf{op}$ to $\widehat{\mathsf{S}}_-$.
4 *Apply* RPNI on $\widehat{\mathsf{S}}$: Run RPNI on the sample set $\widehat{\mathsf{S}}$ over $\widehat{\Sigma}$. Let $\widehat{D} \leftarrow \mathsf{RPNI}(\widehat{\mathsf{S}}; \widehat{\Sigma})$.
5 Construct a DROCA $\mathcal{A}$ from $\widehat{D}$ using the Procedure ConstructOCA.
6 **return** $\mathcal{A}$.

OPNI algorithm. We now introduce OPNI, a passive learning algorithm for DROCA, that uses RPNI as a subroutine. A pseudo-code of the algorithm is given in Algorithm 2. The algorithm can be broadly divided into two major steps: (1) a preprocessing step that creates an enriched sample set (lines 1-3, Algorithm 2), and (2) the inference step that constructs a DROCA consistent with the sample set with the help of RPNI.

In the preprocessing step, we incorporate the counter information about prefixes of S to construct an input $\widehat{\mathsf{S}}$ for RPNI. To that end, we first replace every word u in S with its encoding $\mathsf{Enc}(u)$ (line 1). Let us denote by $\mathsf{Enc}(\mathsf{S})$ the set $\{\mathsf{Enc}(u) \mid u \in \mathsf{S}\}$. We then consider the set $\Sigma_{\mathsf{Act}} = \{\mathsf{Act}(w) \mid w \in \mathsf{Pref}(\mathsf{S})\}$, and the modified alphabet $\widehat{\Sigma} = \widetilde{\Sigma} \cup \Sigma_{\mathsf{Act}}$ (line 2). Recall that, for an alphabet Σ, the alphabet $\widetilde{\Sigma}$ is defined as the set $\bigcup_{\sigma \in \Sigma}\{\sigma^0, \sigma^1\}$. Finally, for every prefix $w \in \mathsf{Pref}(\mathsf{S})$, we add the word $\mathsf{Enc}(w) \cdot \mathsf{Act}(w)$ to $\widehat{\mathsf{S}}_+$, and for every other $\mathsf{op} \in \Sigma_{\mathsf{Act}}$ such that $\mathsf{op} \not\sim \mathsf{Act}(w)$, we add the word $\mathsf{Enc}(w) \cdot \mathsf{op}$ to $\widehat{\mathsf{S}}_-$ (line 3).

In the inference step, we run RPNI on the sample set $\widehat{\mathsf{S}}$ over the alphabet $\widehat{\Sigma}$ (line 4). Let $\widehat{D}$ denote the output DFA of RPNI, then we construct a DROCA $\mathcal{A}$ from $\widehat{D}$ using the construction given in the Procedure ConstructOCA (line 5).

Henceforth, we fix a $\Sigma, \mathsf{S}, \mathsf{ce}$ and $\widehat{D}$. In the remainder of this section, we present the Procedure ConstructOCA and prove the correctness of the OPNI algorithm. We start with the following lemmas (Lemmas 4 and 5) about the DFA $\widehat{D}$.

▶ **Lemma 4.** *The DFA $\widehat{D}$ satisfies the following two properties:*
1. *for all words $w \in \mathsf{S}_+$ (resp. $w \in \mathsf{S}_-$), we have $\mathsf{Enc}(w) \in \mathcal{L}(\widehat{D})$ (resp. $\mathsf{Enc}(w) \notin \mathcal{L}(\widehat{D})$),*
2. *for any two words $w_1, w_2 \in \mathsf{Pref}(S)$, if the runs on $\mathsf{Enc}(w_1)$ and $\mathsf{Enc}(w_2)$ reach the same state in $\widehat{D}$, then $\mathsf{Act}(w_1) \sim \mathsf{Act}(w_2)$.*

Proof.
1. From line 1 of Algorithm 2, we have that, for all $w \in \mathsf{S}_+$ (resp. $w \in \mathsf{S}_-$), $\mathsf{Enc}(w) \in \widehat{\mathsf{S}}_+$ (resp. $\mathsf{Enc}(w) \in \widehat{\mathsf{S}}_-$). Then the property holds from the correctness of RPNI.
2. Assume, towards a contradiction, that there exist words $w_1, w_2 \in \mathsf{S}$, such that the runs on $\mathsf{Enc}(w_1)$ and $\mathsf{Enc}(w_2)$ reach the same state in $\widehat{D}$, but $\mathsf{Act}(w_1) \not\sim \mathsf{Act}(w_2)$. From line 3 of Algorithm 2, we have that, $\mathsf{Enc}(w_1) \cdot \mathsf{Act}(w_1) \in \widehat{\mathsf{S}}_+$, and $\mathsf{Enc}(w_2) \cdot \mathsf{Act}(w_1) \in \widehat{\mathsf{S}}_-$. Then due to the correctness of RPNI, it follows that $\mathsf{Enc}(w_1) \cdot \mathsf{Act}(w_1) \in \mathcal{L}(\widehat{D})$, and $\mathsf{Enc}(w_2) \cdot \mathsf{Act}(w_1) \notin \mathcal{L}(\widehat{D})$. Now since $\mathsf{Enc}(w_1)$ and $\mathsf{Enc}(w_2)$ reaches the same state in $\widehat{D}$, it must also be the case that, $\mathsf{Enc}(w_2) \cdot \mathsf{Act}(w_1) \in \mathcal{L}(\widehat{D})$, which is a contradiction. ◀

▶ **Lemma 5.** *For all $q, q' \in Q$ and $\sigma \in \Sigma$ such that $q \xrightarrow{\sigma^0} q'$ (resp., $q \xrightarrow{\sigma^1} q'$) is a transition in $\widehat{D}$, there must exist $\mathsf{act} \in \Sigma_{\mathsf{Act}}$ and $q_f \in Q$ such that $\mathsf{act}[0] = 0$ (resp., $\mathsf{act}[0] = 1$) and $q \xrightarrow{\mathsf{act}} q_f$. Furthermore, $\mathsf{act}|_\sigma \in \{0, +1\}$ (resp., $\mathsf{act}|_\sigma \in \{-1, 0, +1\}$).*

Proof. Let $q, q' \in Q$ and $\sigma \in \Sigma$ be such that $q \xrightarrow{\sigma^0} q'$. Then, there exists $w\sigma \in \mathsf{Pref}(\mathsf{S})$ such that $q_0 \xrightarrow{\mathsf{Enc}(w)} q$ in $\widehat{D}$ with $\mathsf{ce}(w) = 0$ and $\mathsf{Act}(w)[0] = 0 \in \Sigma_{\mathsf{Act}}$. The latter condition implies that $\mathsf{Enc}(w) \cdot \mathsf{Act}(w) \in \widehat{\mathsf{S}}_+$. Let $\mathsf{act} = \mathsf{Act}(w)$. Therefore, from the correctness of RPNI, we have that $\mathsf{Enc}(w) \cdot \mathsf{act} \in \mathcal{L}(\widehat{D})$, which implies that there exists $q_f \in Q$ such that $q \xrightarrow{\mathsf{act}} q_f$ in $\widehat{D}$. Since $w\sigma$ is in $\mathsf{Pref}(\mathsf{S})$, and $\mathsf{ce}(w\sigma)$ is given in the input, we have that $\mathsf{act}|_\sigma \in \{0, +1\}$. The proof for the other case, when the counter-effect of w is positive, is similar. ◀

Procedure ConstructOCA. Let $\widehat{D} = (Q, \widehat{\Sigma}, q_0, \delta, F)$. We then define the DROCA $\mathcal{A} = (Q, \Sigma, q_0, \delta_0, \delta_1, F)$ where δ_0 and δ_1 are specified as follows. For every $q \in Q$, $\sigma \in \Sigma$, if there is a transition $q \xrightarrow{\sigma^0} q'$ for some q' in $\widehat{D}$, then, from Lemma 5, there exist $\mathsf{act} \in \Sigma_{\mathsf{Act}}$ and $q_f \in Q$ such that $\mathsf{act}[0] = 0$ and $q \xrightarrow{\mathsf{act}} q_f$ with $\mathsf{act}|_\sigma \in \{0, +1\}$. Let $c = \mathsf{act}|_\sigma$. Then define $\delta_0(q, \sigma) = (q', c)$. The definition of δ_1 is analogous.

▶ **Lemma 6.** *Let $\mathcal{A} = \mathit{ConstructOCA}(\widehat{D})$. Then, $\mathcal{A}$ is consistent with the sample set S.*

Proof. We know, from Lemma 4, that for any word $w_1, w_2 \in \Sigma^*$, if $\widehat{D}$ on reading $\mathsf{Enc}(w_1)$ and $\mathsf{Enc}(w_2)$ reaches the same state, then $\mathsf{Act}(w_1)$ is similar to $\mathsf{Act}(w_2)$. This implies that the transition functions of $\mathcal{A}$ are well-defined. We will now show that $\mathcal{A}$ is consistent with the sample S.

▷ Claim 1. For all $w \in \mathsf{Pref}(\mathsf{S})$, $\mathsf{ce}(w) = \mathsf{ce}_{\mathcal{A}}(w)$.

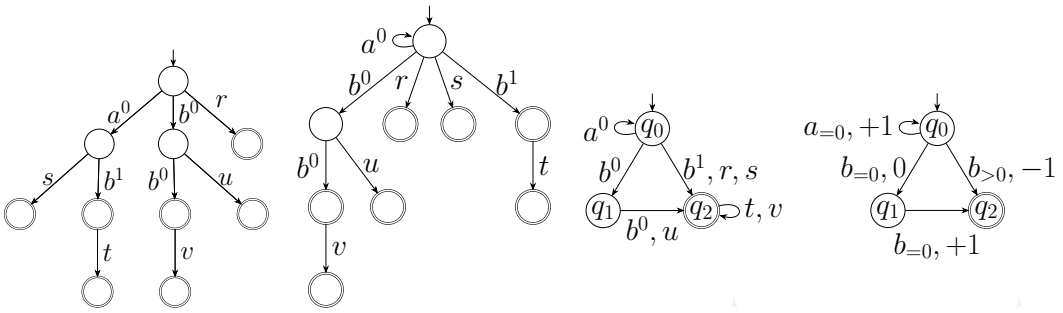
Proof. We will prove this claim by induction on the length of w. This is trivial for $w = \varepsilon$ (base case), since by definition, $\mathsf{ce}(\varepsilon) = \mathsf{ce}_{\mathcal{A}}(\varepsilon) = 0$. Now, assume that the claim is true for all words u of length less than or equal to l, for some $l \geq 0$. Let $u\sigma$ be a prefix in $\mathsf{Pref}(\mathsf{S})$. We will show: $\mathsf{ce}(u\sigma) = \mathsf{ce}_{\mathcal{A}}(u\sigma)$. Let $q, q' \in Q$ be such that $\widehat{D}(\mathsf{Enc}(u)) = q$, and $q \xrightarrow{\sigma^c} q'$ is a transition in $\widehat{D}$, where $c = \mathit{sign}(\mathsf{ce}(u))$. Then, by the construction of $\mathcal{A}$, we have that $\delta_c(q, \sigma) = (q', t)$ where $t = \mathsf{ce}(u\sigma) - \mathsf{ce}(u)$. Therefore, $\mathsf{ce}_{\mathcal{A}}(u\sigma) = \mathsf{ce}_{\mathcal{A}}(u) + t$. Using the induction hypothesis, we can rewrite the above equation as $\mathsf{ce}_{\mathcal{A}}(u\sigma) = \mathsf{ce}(u) + t = \mathsf{ce}(u\sigma)$. This concludes the proof of the claim. ◁

Claim 1 also ensures that $\mathsf{Enc}(w) = \mathsf{Enc}_{\mathcal{A}}(w)$. By construction of $\mathcal{A}$, for all $w \in \mathsf{Pref}(\mathsf{S})$, $\widehat{D}(\mathsf{Enc}(w)) = \mathcal{A}(w)$. Since, from Lemma 4, for all $w \in \mathsf{S}_+$, we have $\mathsf{Enc}(w) \in \mathcal{L}(\widehat{D})$, thus we conclude that $w \in \mathcal{L}(\mathcal{A})$. Similarly for $w \in \mathsf{S}_-$, we have $w \notin \mathcal{L}(\mathcal{A})$. Together with Claim 1, this concludes the proof of the lemma. ◀

Let us now illustrate the OPNI algorithm with an example.

▶ **Example 7.** Let $\mathsf{S} = \mathsf{S}_+ \cup \mathsf{S}_-$ be a sample set over an alphabet $\Sigma = \{a, b\}$ with $\mathsf{S}_+ = \{ab, bb\}$ and $\mathsf{S}_- = \{a, b\}$. Suppose the counter-values for words in $\mathsf{Pref}(\mathsf{S})$ are: $\mathsf{ce} = \{\varepsilon \to 0, a \to 1, b \to 0, ab \to 0, bb \to 1\}$. Below, we illustrate different steps of OPNI on this example.

- Using the Enc function, construct (line 1) the modified sample set $\widehat{\mathsf{S}} = \widehat{\mathsf{S}}_+ \cup \widehat{\mathsf{S}}_-$ where $\widehat{\mathsf{S}}_+ = \{a^0b^1, b^0b^0\}$ and $\widehat{\mathsf{S}}_- = \{a^0, b^0\}$ over the alphabet $\widehat{\Sigma} = \{a^0, a^1, b^0, b^1\}$.
- Then, we construct (line 2) the set Σ_{Act} as follows. We evaluate the Act function for words in $\mathsf{Pref}(S)$. For example, $\mathsf{Act}(\varepsilon) = (\mathit{sign}(\mathsf{ce}(\varepsilon)), \mathsf{ce}(a) - \mathsf{ce}(\varepsilon), \mathsf{ce}(b) - \mathsf{ce}(\varepsilon)) = (0, +1, 0)$. Similarly, we compute $\mathsf{Act}(w)$ for words in $\mathsf{Pref}(\mathsf{S})$, and get $\mathsf{Act} = \{\varepsilon \to$



(a) Prefix Tree. **(b)** Merge(PT;q_ε, q_{a^0}). **(c)** Output of RPNI. **(d)** Output DROCA.

Figure 1 Different steps of the OPNI algorithm on Example 7. Figure 1a represents the prefix tree PT; Figure 1b represents the DFA after merging q_ε and q_{a^0}; Figure 1c is the DFA that is the output of RPNI; Figure 1d is the DROCA output by the OPNI algorithm.

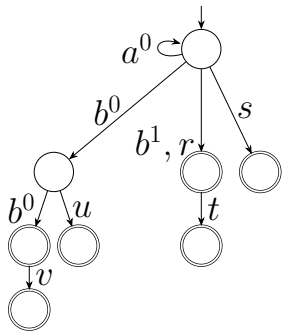
$(0,+1,0), a \to (1,\mathtt{x},-1), ab \to (0,\mathtt{x},\mathtt{x}), b \to (0,\mathtt{x},+1), bb \to (1,\mathtt{x},\mathtt{x})\}$. For simplicity and better readability, we assign symbols from the English alphabet to these action tuples, and write: $(0,+1,0) = r$, $(1,\mathtt{x},-1) = s$, $(0,\mathtt{x},\mathtt{x}) = t$, $(0,\mathtt{x},+1) = u$, and $(1,\mathtt{x},\mathtt{x}) = v$ and therefore, $\Sigma_{\mathsf{Act}} = \{r,s,t,u,v\}$. Consequently, $\widehat{\Sigma} = \{a^0, a^1, b^0, b^1, r, s, t, u, v\}$.

- We then enrich the sample set $\widehat{\mathsf{S}} = \widehat{\mathsf{S}}_+ \cup \widehat{\mathsf{S}}_-$ as in line 3 of the algorithm. For example, since ε is a prefix of S, we will include $\mathsf{Enc}(\varepsilon) \cdot \mathsf{Act}(\varepsilon) = r$ in $\widehat{\mathsf{S}}_+$, and since u is not similar to r, we include u into $\widehat{\mathsf{S}}_-$. Doing this for other elements in $\widehat{\mathsf{S}}$, we get the following sets: $\widehat{\mathsf{S}}_+ = \{a^0b^1, b^0b^0\} \cup \{r, a^0s, a^0b^1t, b^0u, b^0b^0v\}$, and $\widehat{\mathsf{S}}_- = \{a^0, b^0\} \cup \{u, b^0r\}$.
- Next, we apply RPNI (line 4) to this newly constructed sample set $\widehat{\mathsf{S}}$ over the alphabet $\widehat{\Sigma}$. Different steps of the RPNI algorithm on $\widehat{\mathsf{S}}$ are shown in Figure 1. The first two steps are depicted in Figure 1a and Figure 1b, and Figure 1c represents the output of RPNI on $\widehat{\mathsf{S}}$. The intermediate steps of RPNI are shown in Figure 2.
- Finally, we construct a DROCA $\mathcal{A}$ from $\widehat{D}$ using the Procedure ConstructOCA. For example, consider the transition $q_0 \xrightarrow{b^1} q_2$ in $\widehat{D}$. Due to Lemma 5, there must exist a transition from its source state q_0 on an action act with $\mathsf{act}[0] = 1$. We deduce that $\mathsf{act} = r = (1,\mathtt{x},-1)$, and the transition is $q_0 \xrightarrow{r} q_2$. Therefore, according to ConstructOCA procedure, we assign $\delta_1(q_0, b) = (q_2, -1)$. Figure 1d represents the output DROCA of ConstructOCA on $\widehat{D}$.

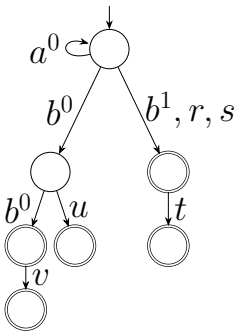
Justification of counter values in the input. We make use of the counter information in the preprocessing step to construct two sets: $\mathsf{Enc}(\mathsf{S})$ and Σ_{Act}. The counter-actions and signs of counter values in the input are required to prevent RPNI on the sample set $\widehat{\mathsf{S}}$ (line 4) from doing some of the merges. Let us illustrate this with an example. Suppose in a certain iteration of RPNI, we are trying to merge two states $D(u)$ and $D(v)$, for some prefixes u and v of S_+, and let both states have outgoing transitions on a letter a. As such, if there is no information on the counter-actions, RPNI will merge these two states, provided that the merged automaton is consistent with S_-. However, suppose that $sign(\mathsf{ce}(u)) = sign(\mathsf{ce}(v))$, but $\mathsf{Act}(u) \not\sim \mathsf{Act}(v)$. In that case, these two states should not be merged, since in any DROCA that is consistent with the sample, u and v must lead to two different states. This justifies the construction of Σ_{Act}. Now suppose that $sign(\mathsf{ce}(u)) \neq sign(\mathsf{ce}(v))$. In that case, those two states could very well be merged. This justifies the preprocessing steps in OPNI.

Since RPNI is guaranteed to terminate, together with Lemma 6, we conclude the following.

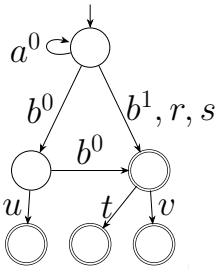
▶ **Theorem 8.** *Algorithm 2 terminates and returns a DROCA consistent with the input.*



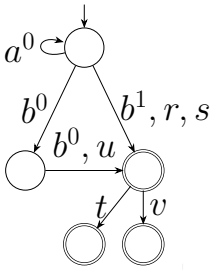
(a) Merge of a^0b^1, r.



(b) Merge of a^0s, r.



(c) Merge of b^0b^0, r.



(d) Merge of b^0u, r.

Figure 2 Intermediate steps of the OPNI algorithm on Example 7.

4 OCA-L*: OPNI-guided active learning procedure for OCA

In this section, we provide a procedure for active learning of DROCA using OPNI.

Learning framework. Following [11, 5], in an *active learning* framework for DROCA, the *learner*'s aim is to construct a DROCA $\mathcal{B}$ by interacting with a *teacher* that has a DROCA $\mathcal{A}$ in mind, such that $\mathcal{A}$ and $\mathcal{B}$ are equivalent. In the process, the learner can ask the teacher three types of queries:

- *Membership queries* $\mathsf{MQ}_{\mathcal{A}}$: the learner provides a word $w \in \Sigma^*$. The teacher returns 1 if $w \in \mathcal{L}(\mathcal{A})$, and 0 if $w \notin \mathcal{L}(\mathcal{A})$.
- *Counter value queries* $\mathsf{CV}_{\mathcal{A}}$: the learner asks the counter value reached on reading a word w. The teacher returns the counter value, *i.e.*, $\mathsf{ce}_{\mathcal{A}}(w)$.
- *Minimal synchronous-equivalence queries* $\mathsf{MSQ}_{\mathcal{A}}$: the learner asks whether a DROCA $\mathcal{C}$ is equivalent and counter-synchronous to $\mathcal{A}$. The teacher returns *yes* only if $\mathcal{C}$ and $\mathcal{A}$ are counter-synchronous and equivalent. Otherwise, the teacher provides a counter-example $z \in \Sigma^*$ with the smallest length such that either $\mathcal{C}(z) \neq \mathcal{A}(z)$ or $\mathsf{ce}_{\mathcal{A}}(z) \neq \mathsf{ce}_{\mathcal{C}}(z)$.

Note that the minimal synchronous-equivalence queries (introduced in [11]) may return a word for which the two machines reach different counter values. This query enables the active learning method to construct a DROCA that is counter-synchronous and equivalent to the teacher's DROCA. As noted in [11], synchronous equivalence is significantly faster than the impractical general equivalence check. Hence, relaxing this condition and having minimal equivalence queries will require the teacher to use a general equivalence check. Thus, in our implementation (described in Section 5), we opt for equivalence queries on minimal counter-synchronous DROCA, as they are faster.

Learning of DROCA

We introduce an active learning procedure for DROCA, called OCA-L*. The pseudocode is presented in Procedure 3, and its key components are detailed below. OCA-L* builds on Angluin's L^* algorithm with modifications inspired by the MinOCA framework from [11].

Observation table. Consider an input alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ for some $k \in \mathbb{N}$. The learner maintains an observation table $T = (\mathsf{R}, \mathsf{C}, Memb, \mathsf{ce}\restriction_{\mathsf{R}\cup\mathsf{R}\Sigma}, \mathsf{Act})$ over Σ. Here R is a nonempty prefix-closed set of strings, C is a nonempty suffix-closed set of strings, $Memb : (\mathsf{R} \cup \mathsf{R}\Sigma)\mathsf{C} \to \{0, 1\}$ is a function that indicates whether a word belongs to the language, $\mathsf{ce}\restriction_{\mathsf{R}\cup\mathsf{R}\Sigma}$: $\mathsf{R} \cup \mathsf{R}\Sigma \to \mathbb{N}$ is the function ce with domain restricted to the set $\mathsf{R} \cup \mathsf{R}\Sigma$, and $\mathsf{Act} : (\mathsf{R} \cup \mathsf{R}\Sigma)\mathsf{C} \to \{0, 1\} \times \{0, 1, -1\}^k$ is a function representing the sign of the counter value reached and the counter-actions on every letter after reading a word. The function Act is the same as the one defined in Section 3. Given $w \in (\mathsf{R}\cup\mathsf{R}\Sigma)\mathsf{C}$, the function $Memb(w)$ returns 0 (resp., 1) if $\mathcal{A}(w)$ is equal to 0 (resp., 1). The observation table initially has $\mathsf{R} = \mathsf{C} = \{\epsilon\}$ and is updated as the procedure runs. Let $\mathsf{C} = \{c_1, \ldots, c_\ell\}$ for some $\ell \in \mathbb{N}$. For any $r \in \mathsf{R} \cup \mathsf{R}\Sigma$, we use $row(r)$ to denote the tuple $(\mathsf{ce}(r), (Memb(rc_1), \mathsf{Act}(rc_1)), \ldots, (Memb(rc_\ell), \mathsf{Act}(rc_\ell)))$.

▶ **Definition 9** (Definition 5 in [11]). *Let $d \in \mathbb{N}$ and $T = (\mathsf{R}, \mathsf{C}, Memb, \mathsf{ce}\restriction_{\mathsf{R}\cup\mathsf{R}\Sigma}, \mathsf{Act})$ be an observation table.*

1. *T is said to be d-*closed *if for all $r' \in \mathsf{R}\Sigma$ with $\mathsf{ce}(r') \leq d$ there exists $r \in \mathsf{R}$ such that $row(r) = row(r')$.*
2. *T is said to be d-*consistent *if for all $r, s \in \mathsf{R}$, such that $\mathsf{ce}(r) = \mathsf{ce}(s) \leq d$ and $row(r) = row(s)$ implies that for all $\sigma \in \Sigma$, $row(r\sigma) = row(s\sigma)$.*

As described in Procedure 3 (see also [11]), the observation table is iteratively expanded until it satisfies the conditions of d-closure and d-consistency. The set of words $w \in (\mathsf{R}\cup\mathsf{R}\Sigma)\mathsf{C}$ is partitioned into S_+ and S_-, based on whether each word is accepted or rejected by the teacher. The goal is to construct a DROCA that solves the corresponding passive learning task: separating S_+ from S_- while being counter-synchronous with the teacher's DROCA for all prefixes of words in $\mathsf{S}_+ \cup \mathsf{S}_-$. This task is delegated to the passive learning algorithm OPNI, which returns a DROCA consistent with the observed data. A DROCA and an observation table corresponding to it are given in Figure 5 and Figure 6 respectively (see Appendix A).

The learner then queries the teacher to verify whether the hypothesis DROCA accepts the same language as the target. If not, the teacher provides a counterexample, which is used to update the observation table. This process repeats until the learner synthesises a DROCA that is equivalent to the teacher's.

It is instructive to contrast our approach with the MinOCA algorithm [11]. While MinOCA uses a SAT solver to solve the passive learning problem and guarantees a minimal separating DROCA, the use of OPNI does not provide any bound on the size of the returned automaton. As a result, OPNI may produce increasingly large automata, and the overall learning procedure OCA-L* may, in principle, fail to terminate. On the other hand, reliance on SAT solvers limits the scalability of MinOCA[2]. In the next section, we show that our approach, based on OPNI, significantly improves scalability compared to MinOCA.

[2] This approach uses the state-of-the-art tool DFAMiner [6] which in turn uses a SAT solver to find the minimal separating DFA.

Procedure 3 OCA-L* for DROCA.

Input: Access to membership ($\mathsf{MQ}_{\mathcal{A}}$), counter value ($\mathsf{CV}_{\mathcal{A}}$), and equivalence ($\mathsf{MSQ}_{\mathcal{A}}$) queries.
Output: A DROCA $\mathcal{B}$ accepting the same language as $\mathcal{A}$.

Initialise $\mathsf{R} \leftarrow \{\varepsilon\}$, $\mathsf{C} \leftarrow \{\varepsilon\}$, $d \leftarrow 0$.
Initialise the observation table $T = (\mathsf{R}, \mathsf{C}, Memb, \mathsf{ce}\restriction_{\mathsf{R}\cup\mathsf{R}\Sigma}, \mathsf{Act})$.
repeat
  while *T is not d-closed or not d-consistent* **do**
    if *T is not d-closed* **then**
      Find $r \in \mathsf{R}$, $a \in \Sigma$ such that $\mathsf{ce}(ra) \leq d$, $row(ra) \neq row(r')$ for all $r' \in \mathsf{R}$.
      Add ra to R.
    if *T is not d-consistent* **then**
      Find $r, s \in \mathsf{R}$, $a \in \Sigma$, $c \in \mathsf{C}$ such that $\mathsf{ce}(r) = \mathsf{ce}(s) \leq d$, $row(r) = row(s)$, and $(Memb(rac) \neq Memb(sac)$ or $\mathsf{Act}(rac) \neq \mathsf{Act}(sac))$.
      Add ac to C.
    Extend $Memb$ and Act to $(\mathsf{R} \cup \mathsf{R}\Sigma)\mathsf{C}$, using membership and counter value queries.
  $\mathsf{S}_+ = \{w \in (\mathsf{R} \cup \mathsf{R}\Sigma)\mathsf{C} \mid Memb(w) = 1\}$.
  $\mathsf{S}_- = \{w \in (\mathsf{R} \cup \mathsf{R}\Sigma)\mathsf{C} \mid Memb(w) = 0\}$.
  $\mathcal{B} \leftarrow \mathsf{OPNI}(\mathsf{S}_+ \cup \mathsf{S}_-, \mathsf{ce}\restriction_{(\mathsf{R}\cup\mathsf{R}\Sigma)\mathsf{C}}; \Sigma)$.
  Ask equivalence query $\mathsf{MSQ}_{\mathcal{A}}(\mathcal{B})$.
  if *teacher gives counter-example z* **then**
    Add all prefixes of z to R.
    Update $d \leftarrow \max\{\max(d, \mathsf{ce}_{\mathcal{A}}(z')) \mid z' \text{ is a prefix of } z\}$.
until *teacher replies yes to an equivalence query*;
return $\mathcal{B}$.

Learning of VOCA

We customise the OCA-L* method for learning VOCA as well. In this setting, the counter-actions are determined by the input alphabet and can therefore be inferred directly from the word. Consequently, the function Act does not have to be included in the observation table. This simplifies the learning process, as the sets S_+ and S_- constructed by OCA-L* are considerably smaller. Furthermore, there are words that do not have a valid run and are denoted by $\mathtt{x}$ in the observation table. The notion of d-closed and d-consistent has to be modified such that an $\mathtt{x}$-marked cell does not have to be compared with a cell in the same column. Moreover, the equivalence check for VOCA is more efficient than for DROCA. A VOCA and an observation table corresponding to it are given in Figure 7 and Figure 8 respectively (see Appendix A).

5 Experimental evaluation

Bruyère et al. [5], were the first to provide a tool for learning DROCA. They employ counter value queries in addition to standard membership and equivalence queries. While effective in principle, this approach does not scale well as the size of the DROCA being learnt increases beyond 7 states.

To address these limitations, Mathew et al. [11] introduced MinOCA, a learning tool capable of inferring DROCA with at most 15 states using a SAT-based approach. Although MinOCA improves upon earlier implementations, its reliance on SAT solvers leads to a significant increase in computational overhead as the size of the automaton grows. In practice, this results in a bottleneck that makes learning larger MinOCA infeasible.

In this work, we present a new approach – OCA-L*– which replaces the SAT-based inference in MinOCA with OPNI (see Section 3). This scales effectively with input size and demonstrates the ability to learn larger DROCA, significantly outperforming existing tools.

We implemented OCA-L* in Python and tested it on randomly generated DROCA. We also implemented a faster learning algorithm for the special case of VOCA. The implementation details and the results obtained are discussed in this section.

Equivalence query. The equivalence of DROCA is known to be in polynomial time [4]. However, as pointed out in [11, 5], the polynomials involved are not suitable for practical applications. Mathew et al. [11] give a practical algorithm for checking the equivalence of VOCA and counter-synchronous DROCA that returns the minimal counter-example. We use their ideas in checking the equivalence of the learnt automaton in our implementation.

Generation of random benchmarks. We generated two datasets for evaluating the performance of the proposed method.

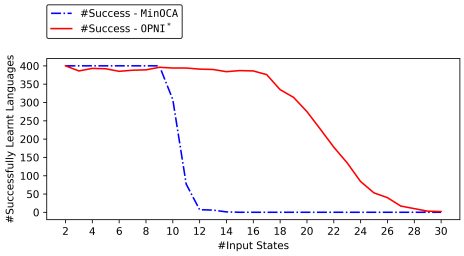
- $\texttt{Dataset}_1$: Random DROCA to compare the performance of OCA-L* with MinOCA.
- $\texttt{Dataset}_2$: Random VOCA to evaluate the performance of OCA-L* for learning VOCA and a similar adaptation of the MinOCA algorithm.

1. Generating random DROCA ($\texttt{Dataset}_1$). In [11], experiments were carried out on randomly generated DROCA. We follow the same procedure to compare our experimental results with those of [11].

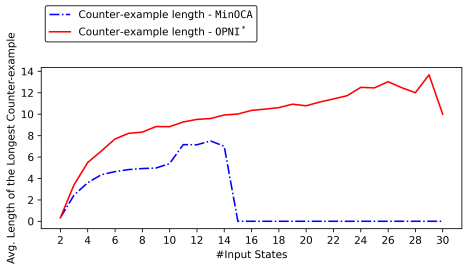
The procedure is explained here for the sake of completeness. Let $n \in \mathbb{N}$ be the number of states of the DROCA to be generated. First, we initialise the set of states $Q = \{q_1, q_2, \ldots, q_n\}$. For all $q \in Q$, we add q to the set of final states F with probability 0.5. If $Q = F$ or $F = \emptyset$ after this step, then we restart the procedure. Otherwise, for all $q \in Q$ and $a \in \Sigma$, we assign $\delta_0(q, a) = (p, c)$ (resp. $\delta_1(q, a) = (p, c)$), with p a random state in Q and c a random counter operation in $\{0, +1\}$ (resp. $\{0, +1, -1\}$). The constructed DROCA is $\mathcal{A} = (Q, \Sigma, \{q_1\}, \delta_0, \delta_1, F)$. If the number of reachable states of $\mathcal{A}$ from the initial configuration is not n, then we discard $\mathcal{A}$ and restart the whole procedure. Otherwise, we output $\mathcal{A}$.

We generated random DROCA with number of states ranging from 2 to 30 and alphabet size ranging from 2 to 5. A total of 100 DROCA were generated for every pair of alphabet and number of states. We reused the dataset used in [8, 11] for DROCA up to 15 states.

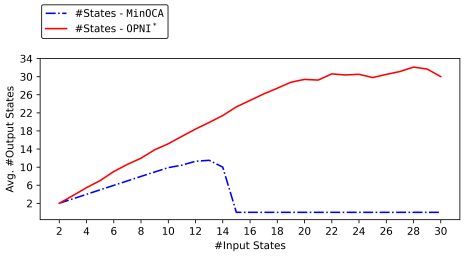
2. Generating random VOCA ($\texttt{Dataset}_2$). For the case of VOCA, we use a similar procedure for generating random VOCA. We generate the states in the same way as above. Following this, for all $a \in \Sigma$, we randomly puts it in one of the sets $\Sigma_{call}, \Sigma_{ret}$, or Σ_{init}. The elements in $\Sigma_{call}, \Sigma_{ret}$ or Σ_{init}, will respectively have $+1, -1$ and 0 as their corresponding counter action. If there does not exist at least one symbol in Σ_{call} and one in Σ_{ret}, then we restart the whole procedure, since the language recognised will be regular. The transitions are also generated as in the case of DROCA. Note that the counter action c in a transition depends on whether the symbol a is in $\Sigma_{call}, \Sigma_{ret}$, or Σ_{init}. The constructed VOCA is $\mathcal{A} = (Q, \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{init}, \{q_1\}, \delta_0, \delta_1, F)$. If the number of reachable states of $\mathcal{A}$ from the initial configuration is not n, then we discard $\mathcal{A}$ and restart the whole procedure. Otherwise, we output $\mathcal{A}$.



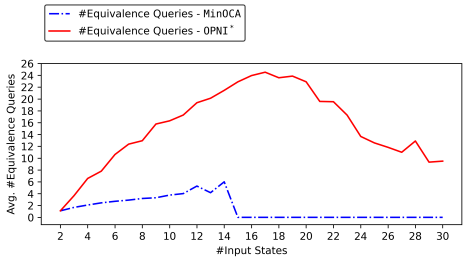
(a) Number of successfully learnt languages by OCA-L* (Out of 400) with 5 minutes timeout.



(b) The average length of the longest counter-example.



(c) Average number of states in the learnt DROCA.



(d) Average number of equivalence queries used.

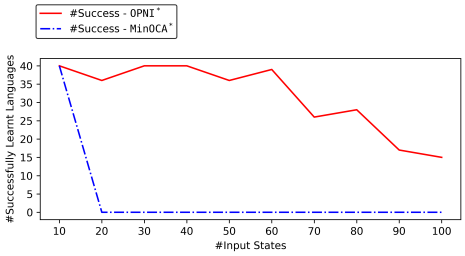
Figure 3 Evaluation of OCA-L* and MinOCA for DROCA on $\texttt{Dataset}_1$.

We generated random VOCA with number of states ranging from 10 to 100 and input alphabet size ranging from 2 to 5. A total of 10 VOCA were generated for every pair of input alphabet and number of states. The number of randomly generated VOCA was limited to 10 per configuration due to the increasing difficulty in constructing a valid automaton where all states are reachable and satisfy the remaining conditions. Moreover, our primary objective is to evaluate how well the proposed approach scales with input size. For this purpose, the selected sample size is sufficient to observe meaningful trends in performance without incurring excessive computational overhead.

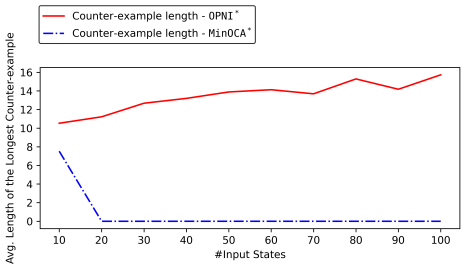
Experimental results. All the experiments were performed on an Apple M1 chip with 8GB of RAM. We implemented the proposed method (OCA-L*) in Python for DROCA and then for the special case of VOCA[3] In Figure 3a (resp. Figure 4a), the total number of languages is out of 400 (resp. 40) for each input size because, for every number of states, we generated 400 (resp. 40) random DROCA (resp. VOCA) for input alphabet sizes 2, 3, 4, and 5; to keep the visualization simple, we omitted the z-axis for alphabet size and instead aggregated the number of successfully learnt languages across all alphabet sizes for each input size – this approach is used consistently across all graphs.

1. Evaluating the performance OCA-L and MinOCA on* $\texttt{Dataset}_1$: We first compare the performance of OCA-L* for DROCA with that of MinOCA [11]. A timeout of 5 minutes was allotted for both MinOCA and OCA-L* for learning each DROCA. If the procedure times out,

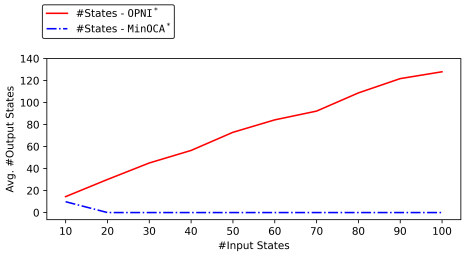
[3] The implementations of OCA-L* for DROCA, OCA-L* and MinOCA for VOCA, the datasets used, and the complete results generated are available at [9]. The implementation of MinOCA for DROCA is provided in [8].



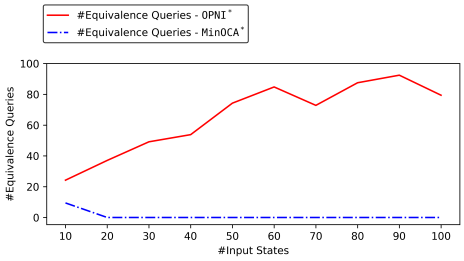
(a) Number of successfully learnt languages (Out of 40) with a timeout of 20 minutes.



(b) The average length of the longest counter-example.



(c) Average number of states in the learnt VOCA.



(d) Number of equivalence queries used.

Figure 4 Evaluation of OCA-L* and MinOCA for VOCA on $\texttt{Dataset}_2$.

we discard that input and process the next one. The number of languages successfully learnt by OCA-L* and MinOCA for different input sizes is depicted in Figure 3a. The proposed method outperforms MinOCA in terms of the number of successfully learnt languages within the given timeout as the number of states increases. The averages presented in the remaining graphs are computed using only those languages that were successfully learnt. Note that the y-values for MinOCA remain zero in all graphs for input sizes with more than 15 states, as it fails to learn any language within the specified timeout.

Figure 3b shows the average length of the longest counter-example. Since MinOCA guarantees learning a minimal counter-synchronous DROCA, whereas OCA-L* provides no such guarantee, the DROCA learnt by OCA-L* typically has more states. Consequently, the length of the counterexamples also tends to be longer for OCA-L*. Figure 3c shows the average number of states in the learnt DROCA. MinOCA learns a minimal counter-synchronous DROCA equivalent to the input. However, the DROCA learnt by OPNI is equivalent and counter-synchronous with respect to the input, but not minimal. Figure 3d shows the average number of equivalence queries used for successfully learning the input DROCA. We also add in Appendix B Figure 9a and Figure 9b the average number of rows and columns in the final observation table.

Note that OCA-L* is a semi-algorithm; its termination is not guaranteed. However, our experimental results on randomly generated benchmarks show that it works well in practice.

2. Evaluating the performance OCA-L and MinOCA on* $\texttt{Dataset}_2$: We now compare the performance of OCA-L* for VOCA with that of MinOCA [11]. We note that OCA-L* for VOCA is faster compared to OCA-L* for DROCA. This is mainly due to the following two reasons:

1. faster algorithm for checking equivalence of VOCA, and
2. the input alphabet itself determines the counter actions, and therefore, step 2 and step 3 of Algorithm 2 can be skipped while passive learning VOCA.

A similar modification of MinOCA for VOCA was also implemented. A timeout of 20 minutes was allotted for both procedures for learning each VOCA. If the procedure times out, we discard that input and process the next one. The number of VOCA languages successfully learnt by OCA-L* for different input sizes is shown in Figure 4a. We observe that OCA-L* is able to learn close to 50% of VOCA of 100 states. However, it was not able to learn any VOCA with more than 20 states. The averages presented in the remaining graphs are computed using only those languages that were successfully learnt. Note that the y-values for MinOCA remain zero in all graphs for input sizes with more than 20 states, as it fails to learn any language within the specified timeout.

The results of our experiments are shown in Figure 4. Figure 4b shows the average length of the longest counter-example. Figure 4c shows the average number of states in the learnt VOCA. Figure 4d shows the average number of equivalence queries used for successfully learning the input VOCA. In this case also, the VOCA learnt by MinOCA is minimal, whereas the one learnt by OCA-L* is not. Figure 10a and Figure 10b (see Appendix B) show the average number of rows and columns in the final observation table.

6 Conclusion

This work focuses on passive and active learning of deterministic real-time one-counter automata (DROCA). Inspired by the classical RPNI algorithm, we developed a passive learning algorithm, OPNI, tailored to DROCA. Building on this, we showed how active learning can be guided by a sequence of passive learning tasks, and proposed an active learning method, OCA-L*, which uses OPNI as a subroutine. In contrast, the state-of-the-art MinOCA algorithm employs a SAT solver for this step.

Our experiments demonstrate that OCA-L* scales significantly better than MinOCA in practice. We also used OCA-L* to learn visibly one-counter automata (VOCA), observing that it can successfully learn automata with up to 100 states. Despite its practical advantages, a limitation of OCA-L* is that it does not guarantee termination on all inputs, since OPNI may return increasingly large automata in the absence of minimality guarantees.

A fundamental bottleneck – common to our approach and to prior works such as [5, 11] – is the reliance on counter-value queries. This renders the learning process a grey-box framework, in contrast to Angluin's black-box L^* algorithm for finite automata. Recent work [10] has shown that active learning of DROCA is possible in polynomial time even without counter-value queries. However, the proposed algorithm is not practical due to the high-degree polynomials involved. An interesting direction for future work is to combine insights from the methods: OCA-L*, MinOCA [11], and OL* [10] to develop learning algorithms that are both theoretically efficient and practically usable.

Finally, extending these ideas to richer models such as pushdown automata – or their subclasses, such as visibly pushdown automata – offers a direction for future research.

References

1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

2 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

3 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

4 Stanislav Böhm and Stefan Göller. Language equivalence of deterministic real-time one-counter automata is nl-complete. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 2011. doi:10.1007/978-3-642-22993-0_20.

5 Véronique Bruyère, Guillermo A. Pérez, and Gaëtan Staquet. Learning realtime one-counter automata. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2022. doi:10.1007/978-3-030-99524-9_13.

6 Daniele Dell'Erba, Yong Li, and Sven Schewe. DFAMiner: Mining minimal separating dfas from labelled samples. In *FM (2)*, volume 14934 of *Lecture Notes in Computer Science*, pages 48–66. Springer, 2024. doi:10.1007/978-3-031-71177-0_4.

7 Amr F. Fahmy and Robert S. Roos. Efficient learning of real time one-counter automata. In *ALT*, volume 997 of *Lecture Notes in Computer Science*, pages 25–40. Springer, 1995. doi:10.1007/3-540-60454-5_26.

8 Prince Mathew. MinOCA, January 2025. doi:10.5281/zenodo.14604419.

9 Prince Mathew. OCA-L*, October 2025. doi:10.5281/zenodo.17310292.

10 Prince Mathew, Vincent Penelle, and A. V. Sreejith. Learning deterministic one-counter automata in polynomial time. In *LICS*, pages 444–457, 2025.

11 Prince Mathew, Vincent Penelle, and A. V. Sreejith. Learning real-time one-counter automata using polynomially many queries. In *TACAS (1)*, volume 15696 of *Lecture Notes in Computer Science*, pages 276–294. Springer, 2025. doi:10.1007/978-3-031-90643-5_14.

12 Daniel Neider and Christof Löding. Learning visibly one-counter automata in polynomial time. *Technical Report, RWTH Aachen*, AIB-2010-02, 2010.

13 José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

14 Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975. doi:10.1016/S0022-0000(75)80005-5.

A Section 4. OCA-L*: OPNI-guided active learning procedure for OCA

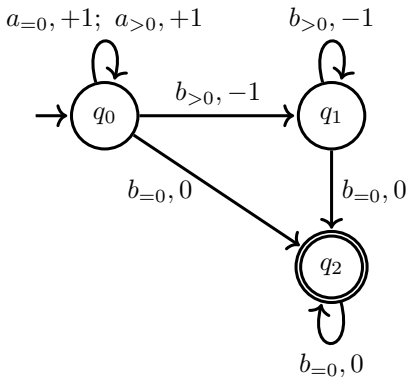


Figure 5 A DROCA recognising the language $\{a^n b^m \mid m > n\}$. Transitions not shown go to a non-final sink state with counter-action 0.

		ce	ϵ	
			Memb	Act
R	ϵ	0	0	$(0, +1, 0)$
	a	1	0	$(1, +1, -1)$
	b	0	1	$(0, 0, 0)$
	ab	0	0	$(0, 0, 0)$
RΣ	aa	2	0	$(1, +1, -1)$
	ba	0	0	$(0, 0, 0)$
	bb	0	1	$(0, 0, 0)$
	aba	0	0	$(0, 0, 0)$
	abb	0	1	$(0, 0, 0)$

Figure 6 Observation table for the DROCA in Figure 5. The table is 1-closed and 1-consistent. Here, $\mathsf{R} = \{\epsilon, a, b\}$ and $\mathsf{C} = \{\epsilon\}$.

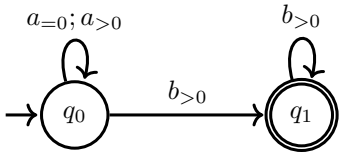
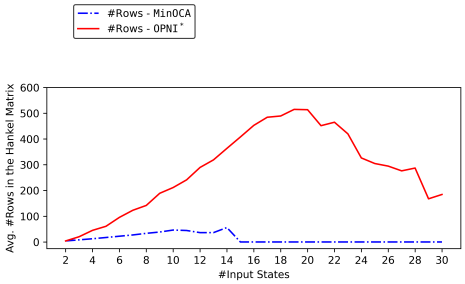


Figure 7 A VOCA over the alphabet $\{a, b\}$, where $\Sigma_{call} = \{a\}$, $\Sigma_{ret} = \{b\}$ and $\Sigma_{int} = \emptyset$. The VOCA recognises the language $\{a^n b^m \mid m \leq n\}$. The transitions not shown go to a non-final sink state.

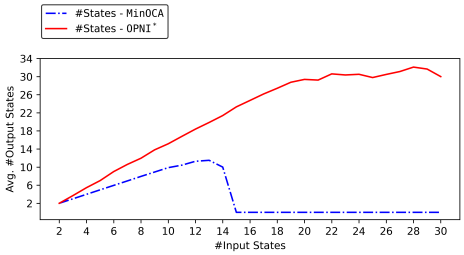
		ce	ϵ
			Memb
R	ϵ	0	0
	a	1	0
	ab	0	1
RΣ	b	x	x
	aa	2	0
	aba	1	0
	abb	x	x

Figure 8 Observation table for the VOCA in Figure 7. The table is both 1-closed and 1-consistent. Here, $\mathsf{R} = \{\epsilon, a, ab\}$ and $\mathsf{C} = \{\epsilon\}$.

B Section 5. Experimental evaluation

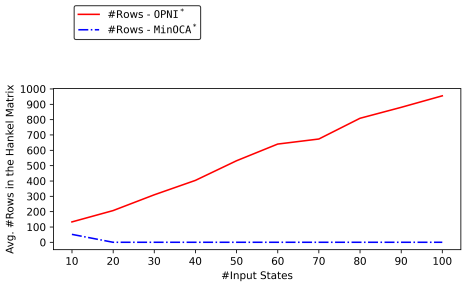


(a) The average number of rows in the observation table.

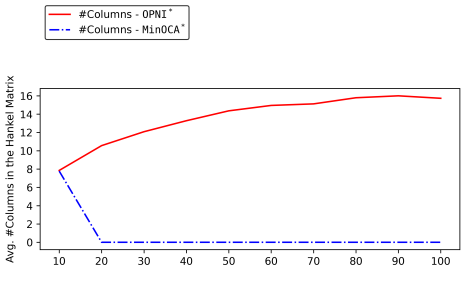


(b) The average number of columns in the observation table.

Figure 9 Evaluation of OCA-L* and MinOCA for DROCA on $\texttt{Dataset}_1$.



(a) The average number of rows in the observation table.



(b) The average number of columns in the observation table.

Figure 10 Evaluation of OCA-L* and MinOCA for VOCA on $\texttt{Dataset}_2$.

Hardness of Finding Kings and Strong Kings

Ziad Ismaili Alaoui
Department of Computer Science, University of Liverpool, UK

Nikhil Mande
Department of Computer Science, University of Liverpool, UK

Abstract

A king in a directed graph is a vertex v such that every other vertex is reachable from v via a path of length at most 2. It is well known that every tournament (a complete graph where each edge has a direction) has at least one king. Our contributions in this work are:

- We show that the query complexity of determining existence of a king in arbitrary n-vertex digraphs is $\Theta(n^2)$. This is in stark contrast to the case where the input is a tournament, where Shen, Sheng, and Wu [SICOMP'03] showed that a king can be found in $O(n^{3/2})$ queries.
- In an attempt to increase the "fairness" in the definition of tournament winners, Ho and Chang [IPL'03] defined a *strong king* to be a king k such that, for every v that dominates k, the number of length-2 paths from k to v is strictly larger than the number of length-2 paths from v to k. We show that the query complexity of finding a strong king in a tournament is $\Theta(n^2)$. This answers a question of Biswas, Jayapaul, Raman, and Satti [DAM'22] in the negative.

A key component in our proofs is the design of specific tournaments where every vertex is a king, and analyzing certain properties of these tournaments. We feel these constructions and properties are independently interesting and may lead to more interesting results about tournament solutions.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Oracles and decision trees; Mathematics of computing → Combinatorics

Keywords and phrases Tournaments, kings, query complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.36

1 Introduction

A tournament is a complete directed graph. Motivated to find a reasonable notion of a "most-dominant chicken" in a flock of chickens (where the "pecking order" is represented as a tournament), Landau [10] introduced the notion of a *king* in a tournament. A king in a tournament is a vertex k such that for every other vertex v, either there is an edge from k to v, or there exists a u such that there are edges from k to u and from u to v. That is, a king is a vertex such that every other vertex is reachable from it via a path of length at most 2. The notion of a king is a natural generalisation of that of a source: a vertex is a source if every other vertex is reachable from it via a path of length 1. While it is easy to see that a tournament need not have a source, it is also not hard to show that every tournament has at least one king [10, 13]. Soon after Maurer's popular science article about kings in tournaments [13], Reid [16] showed the existence of tournaments in which every vertex is a king. Tournament solutions/winners are also a natural object of study in social choice theory. See, for instance, [5] and the references therein. The monograph by Moon [14] sparked a line of research on tournaments, their solutions, and their structural properties.

A natural computational model to study the complexity of finding tournament solutions and related problems is that of *query complexity*. In this setting, an algorithm has access to the input in the form of *queries*: in one step an algorithm may query the direction of the edge between a pair of vertices of its choosing (in case of an undirected graph problem, the algorithm may query the presence/absence of an edge between a pair of vertices of its choosing). The algorithm's goal is to minimize the number of queries made in the worst case,


© Ziad Ismaili Alaoui and Nikhil Mande;
licensed under Creative Commons License CC-BY 4.0

and all other operations are treated as free. There is a rich literature on the study of graph problems in the setting of query complexity, see, for example, [18, 17, 21, 7, 3, 5, 12] and the references therein.

The focus of this paper is on finding a king in a digraph. Given that every tournament has a king as mentioned earlier, it is natural to ask: how efficiently can a king in an n-vertex tournament be found? Note that this is equivalent to determining whether a digraph has a *radius*[1] of at most 2. The study of this question was initiated by Shen, Sheng, and Wu [19], who exhibited an algorithm with query complexity $O(n^{3/2})$, and they also showed a non-matching lower bound of $\Omega(n^{4/3})$. To date, these are the best-known bounds on the complexity of finding a king in a tournament, and closing the gap remains an intriguing open problem. There have been some very recent works showing better query bounds for variants of the task of finding a king, or studying the complexity of finding kings in models other than deterministic query complexity [2, 9, 12, 11, 1]. Our first main contribution in this paper to show that if we allow for non-tournament inputs, then the query complexity of even determining the existence of a king shoots up.

▶ **Theorem 1.** *The randomized query complexity of determining the existence of a king in an n-vertex digraph is $\Theta(n^2)$.*

Relevant to our work is the notion of a *strong king* in a tournament. This notion was defined by Ho and Chang [8] in an attempt to increase the "fairness" of the notion of a winner in a tournament. Let $\delta(u, v)$ denote the number of length-2 paths from u to v. A king k is said to be *strong* if for all w with $w \to k$, we have $\delta(k, w) > \delta(w, k)$. To see why this notion increases the fairness of the definition of a winner, consider the following scenario: say we have a tournament in which a vertex k has a single out-neighbour u, and all other vertices are in-neighbours of k. Further suppose every vertex other than k and u is an out-neighbour of u. The vertex k is a king in this tournament, but not necessarily a "fair" one, since it "defeats" only one vertex. However, it is not hard to see that k is not a strong king. The study of strong kings in tournaments has also gained some attention [4, 15]. Biswas et al. [2, Section 7] asked if one could devise an algorithm to find a strong king in an n-vertex tournament with cost $o(n^2)$. Our second main contribution in this paper is to answer this in the negative.

▶ **Theorem 2.** *The randomized query complexity of finding a strong king in an n-vertex tournament is $\Theta(n^2)$.*

1.1 Organization of the Paper

In Section 2, we review basic preliminaries and formalize the computational questions and models of interest. In Section 3, we recall two known constructions of tournaments where every vertex is a king. In Section 4, we show certain properties of these constructions and use these properties to conclude our main results, Theorem 1 and Theorem 2 (restated formally as Theorem 14 and Theorem 18, respectively). Section 4 contains the main technical novelty of this paper. Finally, in Section 5, we recall our main results and discuss potential directions for future work.

[1] The radius of a directed graph is the minimum eccentricity of any vertex, where the eccentricity of a vertex is defined to be the greatest distance from it to any other vertex.

2 Preliminaries

2.1 Basic Concepts and Notation

For a positive integer n, let $[n] = \{1, 2, \ldots, n\}$ and define $[n]_0 = \{0\} \cup [n-1]$. We use S_n to denote the set of permutations on the elements in $[n]_0$. For non-negative integers n and $i \in [n]_0$, let $\sigma_i \in S_n$ denote the permutation defined by $\sigma_i(j) = (j + i) \pmod n$ for all $j \in [n]_0$. Let G be an n-vertex graph with vertex set $[n]_0$, and let $i \in [n]_0$. Define $\sigma(G, i)$ to be the graph obtained by relabelling each vertex v of G by $\sigma_i(v)$.

▶ **Definition 3** (Tournament, Subgraph, Subtournament). A *tournament* $T = (V, E)$ is a *complete* directed graph where V is the set of vertices, E is the set of directed edges, and for every edge $(u, v) \in E$, $(v, u) \notin E$. A *subgraph* induced by $G = (V, E)$ on $V' \subseteq V$ is a graph $G[V'] = (V', E')$ such that $E' = E \cap \{(u, v) : u \in V' \text{ and } v \in V'\}$. A *subtournament* is analogously defined with respect to tournaments.

We denote $V(G)$ and $E(G)$ as the vertex and edge sets of G, respectively. We often use the notation (u, v) and $u \to v$ interchangeably. For an edge $e = (u, v)$, we say that u is the *source* of e and v is the *target* of e. A vertex $v \in V$ is an *out-neighbor* of $u \in V$ if $(u, v) \in E$, and v is an *in-neighbor* of u if $(v, u) \in E$. We say that a vertex v *dominates* u when $v \to u$. The set of all out-neighbors of a vertex v is denoted by $\Gamma^+(v)$, and the set of all in-neighbors of v is denoted by $\Gamma^-(v)$. Define $d^+(v) = |\Gamma^+(v)|$ and $d^-(v) = |\Gamma^-(v)|$. A (directed) *path* of length n is an ordered sequence of edges $e_1, e_2, \ldots, e_n$ such that the target of e_i is the source of e_{i+1} for all $1 \leq i < n$.

▶ **Definition 4** (King). A *king* in a directed graph is a vertex v such that, for all other vertices $u \neq v$, at least one of the following holds:

- $v \to u$ is an edge in the graph, or
- there exists a vertex w such that $v \to w$ and $w \to u$ are edges in the graph.

In other words, a king is a vertex that can reach every other vertex by a path of at most two edges. It is well known [10] that every tournament has at least one king, and the maximum-out-degree vertex is always a king. A king that dominates all other vertices is called a *source*.

▶ **Lemma 5** ([13, Theorem 7]). *Let $T = (V, E)$ be an arbitrary tournament. T contains exactly one king if and only if that king is a source.*

Define $\delta(u, v)$ as the number of length-2 paths from vertex u to vertex v. A king k is said to be *strong* if, for each vertex v such that $v \to k$, we have $\delta(k, v) > \delta(v, k)$. That is, a king k is said to be strong if, for any vertex v that directly dominates it, there are strictly more length-2 paths from k to v than there are from v to k. Ho and Chang [8, Lemma 1] observed that every tournament has at least one strong king. We include a proof for completeness.

▶ **Lemma 6** ([8, Lemma 1]). *Let $T = (V, E)$ be an arbitrary tournament, and let $u \in V$ be a maximum-out-degree vertex in T. Then, u is a strong king.*

Proof. Towards a contradiction, suppose u is not a strong king. Then, there must exist some vertex v such that $v \to u$ and $\delta(v, u) \geq \delta(u, v)$.

Define $W = \{w \in V \setminus \{u, v\} : u \to w \text{ and } v \to w\}$. Note that $d^+(u) = \delta(u, v) + |W|$ and $d^+(v) = \delta(v, u) + |W| + 1 > d^+(u)$, which is a contradiction since we assumed u to be a maximum-out-degree vertex. ◀

A set of vertices $S \subseteq V$ is a *dominating set* in $T = (V, E)$ if, for every $v \in V$, either $v \in S$ or there exists $w \in S$ such that $w \to v$. A *dominating pair* is a dominating set of exactly two vertices.

2.2 Formal Definitions of Computational Tasks

We represent an n-vertex graph $G = (V = [n]_0, E)$ as a binary string in $\{0,1\}^{\alpha_n}$, where $\alpha_n = \binom{n}{2}$ if G is a tournament, or $\alpha_n = n^2$ otherwise. An element of $[n]_0$ corresponds to the label of a vertex, and there is one variable ($\{i, j\}$) per edge (between vertex i and vertex j) that defines its direction. In a tournament, we assume $\alpha_n = \binom{n}{2}$ because for each pair of vertices, there is exactly one directed edge between them. Define the relation $\mathsf{KING}_n \subseteq \{0,1\}^{\alpha_n} \times [n]_0$ by

$$(G, v) \in \mathsf{KING}_n \text{ if } \forall u \in [n]_0 \setminus \{v\}\,, \text{ either } v \to u \text{ or } \exists w, v \to w \to u,$$

and define $K_n \subseteq \{0,1\}^{\alpha_n}$ as the language of all n-vertex graphs G such that $\exists v \in V$, $(G, v) \in \mathsf{KING}_n$. In other words, K is the language of all n-vertex graphs containing at least one king. Let $\mathsf{EXIST\text{-}KING}_n \subseteq \{0,1\}^{\alpha_n} \times \{0,1\}$ be the relation such that $(G, 1) \in \mathsf{EXIST\text{-}KING}_n$ if and only if $G \in K_n$.[2] Finally, for $v \in V$, define the relation $\mathsf{STRONG\text{-}KING}_n \subseteq \{0,1\}^{\alpha_n} \times [n]_0$ by

$$(G, v) \in \mathsf{STRONG\text{-}KING}_n \text{ if } \forall u \in [n]_0 \text{ such that } u \to v, \delta(v, u) > \delta(u, v).$$

2.3 Deterministic Query Complexity

A deterministic decision tree $\mathcal{T}$ on m variables is a binary tree where the internal nodes are labelled by variables, and leaves are labelled with elements of a set $\mathcal{R}$. Each internal node has a left child, corresponding to an edge labelled 0, and a right child, corresponding to an edge labelled 1. On an input $x \in \{0,1\}^m$, $\mathcal{T}$'s computation traverses a path from root to leaf as follows: At an internal node, the variable associated with that node is *queried*: if the value obtained is 0, the computation moves to the left child, otherwise it moves to the right child. The output of $\mathcal{T}$ on input x, denoted by $\mathcal{T}(x)$, is the label of leaf node reached.

We say that a decision tree $\mathcal{T}$ *computes* the relation $f \subseteq \{0,1\}^m \times \mathcal{R}$ if $(x, \mathcal{T}(x)) \in f$ for all $x \in \{0,1\}^m$. The deterministic query complexity of f, is

$$\mathsf{D}(f) := \min_{\mathcal{T}:\mathcal{T} \text{ computes } f} \text{depth}(\mathcal{T}).$$

That is, $\mathsf{D}(f)$ represents the minimum depth of any deterministic decision tree that computes f.

2.4 Randomized Query Complexity

A randomized decision tree $\mathcal{A}$ is a distribution $\mathcal{D}_\mathcal{A}$ over deterministic decision trees. On input $x \in \{0,1\}^m$, the computation of $\mathcal{A}$ proceeds by first sampling a deterministic decision tree $\mathcal{T}$ according to $\mathcal{D}_\mathcal{A}$, and outputting the label of the leaf reached by $\mathcal{T}$ on x. We say $\mathcal{A}$ computes f to error ε if for every input x, $\Pr[(x, \mathcal{A}(x)) \in f] \geq 1 - \varepsilon$. The randomized query complexity of $f \subseteq \{0,1\}^m \times \mathcal{R}$ is defined as follows:

$$\mathsf{R}_\varepsilon(f) = \min_{\substack{\mathcal{A} \text{ computing } f \\ \text{with error} \leq \varepsilon}} \max_{\mathcal{T}:\mathcal{D}_\mathcal{A}(\mathcal{T})>0} \text{depth}(\mathcal{T}).$$

[2] Note that we could also define $\mathsf{EXIST\text{-}KING}_n$ as a function from $\{0,1\}^{\alpha_n} \to \{0,1\}$ where $\mathsf{EXIST\text{-}KING}_n(G) = 1$ iff G contains a king, but we choose to define it as a relation for the sake of consistency with other definitions.

When we drop the subscript ε, we assume $\varepsilon = 1/3$. We use Yao's minimax principle [20], stated below in a form convenient for us.

▶ **Lemma 7** (Yao's Minimax Principle). *For a relation $f \subseteq \{0,1\}^m \times \mathcal{R}$, we have $\mathsf{R}_\varepsilon(f) \geq k$ if and only if there exists a distribution $\mu : \{0,1\}^m \to [0,1]$ such that $\mathsf{D}_\mu(f) \geq k$. Here, $\mathsf{D}_\mu(f)$ is the minimum depth of a deterministic decision tree that computes f to error at most ε when inputs are drawn from the distribution μ.*

3 Balanced Tournaments

A *balanced* tournament is one in which every vertex is a king. Maurer [13] showed that a balanced tournament exists for every positive integer n, except when $n = 2$ or $n = 4$; this result naturally guarantees the existence of a balanced tournament for every odd n.

In this section, we present two distinct constructions of balanced tournaments. In subsequent sections, we analyze key properties of these constructions that help us prove our bounds. We clarify here that both of these constructions have appeared in prior works, but the properties that we require of them have not been analyzed earlier, to the best of our knowledge.

▶ **Construction 8** ([13, Lemma 7]). Let n be an odd positive integer. Define Δ_n as a tournament on n vertices, labelled by elements of $[n]_0$, recursively constructed as follows:

- If $n = 1$, Δ_n consists of a single vertex labelled 0.
- If $n > 1$, build Δ_{n-2} as a subtournament induced by $[n-2]_0$. Then direct the edges such that:
  1. for each vertex v in $[n-2]_0$, $(n-1) \to v$ and $v \to (n-2)$, and
  2. $(n-2) \to (n-1)$.

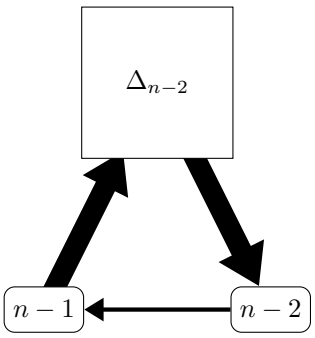


Figure 1 Illustration of Construction 8.

Figure 1 illustrates the construction:[3] vertex $n-1$ is dominated by vertex $n-2$ and dominates all vertices in Δ_{n-2}, and vertex $n-2$ dominates vertex $n-1$ and is dominated by all vertices in Δ_{n-2}. For the rest of this paper, we use Δ_n to denote the n-vertex tournament as defined in Construction 8. While the following was already shown by Maurer [13, Lemma 7], we include a proof for completeness.

▶ **Lemma 9.** *Let n be an odd positive integer. The tournament Δ_n is balanced.*

[3] The shape of the figure may clarify the choice of "Δ".

Proof. Let us proceed by induction on the number of vertices n.

When $n = 1$, Δ_n is clearly a balanced tournament, as its single vertex trivially satisfies the condition of being a king. Indeed, since there are no other vertices, it vacuously reaches all other vertices by a path of length 1 or 2.

Now, suppose the lemma holds for some n, we show that augmenting the tournament by two vertices following the instructions of Construction 8 does not violate the invariant; that is, Δ_{n+2} still remains balanced. By the induction hypothesis, every vertex in $[n]_0$ reaches all other vertices within that set via a path of length 1 or 2. When two new vertices, $n+1$ and n, are added:

- vertex $n+1$ dominates all vertices in $[n]_0$;
- all vertices in $[n]_0$ dominate n; and
- vertex n dominates $n+1$.

Clearly, none of the vertices in $[n]_0$ loses its status as a king, as each vertex in $[n]_0$ reaches n directly and reaches $n+1$ via n. Furthermore, the newly added vertices also satisfy the king condition:

- vertex $n+1$ reaches n via one of the vertices in $[n]_0$, and n reaches $n+1$ directly; and
- both $n+1$ and n reach all vertices in $[n]_0$ (either directly or via n).

Thus, the property is preserved and Δ_{n+2} remains balanced. ◀

The following construction has appeared in the literature as an example of a *rotational tournament* [6, Section 2].

▶ **Construction 10.** Let n be an odd positive integer. Define U_n as a tournament on n vertices, labelled by elements of $[n]_0$. For each pair of vertices i, j with $i < j$, direct the edge as follows:

- If $i + j$ is odd, direct the edge from i to j ($i \to j$).
- Otherwise, direct the edge from j to i ($j \to i$).

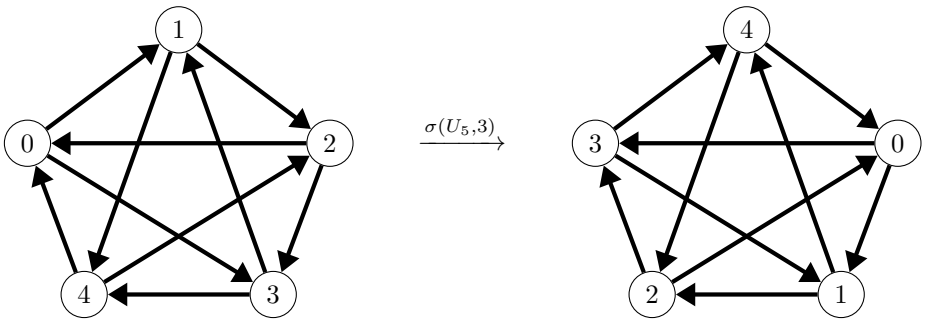


Figure 2 The tournament U_5 and the tournament $\sigma(U_5, 3) \in \mathrm{Aut}(U_5)$ obtained by rotating the labels cyclically clockwise by 3 positions.

An example is depicted in Figure 2 when $n = 5$. While this implicitly follows from the work of [6], we now show from first principles that U_n is balanced for any odd positive integer n. Again, for the rest of this paper, we use U_n to denote the n-vertex tournament as defined in Construction 10.

▶ **Lemma 11.** *Let n be an odd positive integer. The tournament U_n is balanced.*

One way to prove Lemma 11 is to show that each vertex in U_n (for an odd positive n) dominates exactly $\frac{n-1}{2}$ vertices. Thus, given that a maximum-out-degree vertex in a tournament is a king [10], it follows that all vertices are kings. For completeness, we include a proof from first principles below.

Proof of Lemma 11. Observe that each vertex i dominates all lesser vertices of similar parity and all greater vertices of opposite parity. Define the parity function $p(x) = x \pmod 2$; formally, we have $\Gamma^+(i) = \{j \in [n] : j < i \text{ and } p(j) = p(i)\} \cup \{j \in [n] : j > i \text{ and } p(i) \neq p(j)\}$.

Consider any vertex $i < n-1$. Since $p(i+1) \neq p(i)$, vertex i dominates $i+1$. Thus, $i+1$ in turn dominates all vertices less than i of opposite parity and all greater vertices of the same parity; formally, $\Gamma^-(i) = \Gamma^+(i+1)$. Hence, i reaches all vertices by a path of at most two edges either through direct domination, or via vertex $i+1$.

For $i = n-1$, note that $n-1$ is even, so it dominates all even vertices, including vertex 0. Since vertex 0 dominates all odd vertices, and $n-1$ dominates 0, it follows that $n-1$ is also a king. Therefore, every vertex in U_n is a king, and the tournament is balanced. ◀

An interesting property of U_n is its symmetry. Indeed, observe that rotating the tournament (i.e., shifting its labels) clockwise or anti-clockwise results in the same labelled tournament. This is useful as it allows a property of one vertex to be extended to all vertices in U_n.

▶ **Lemma 12** ([6]). *Let n be an odd positive integer. For $i \in \mathbb{Z}$, we have $\sigma(U_n, i) \in \mathrm{Aut}(U_n)$.*

4 Lower Bounds

In this section, we prove our two main results: determining the existence of a king in an arbitrary n-vertex digraph (Subsection 4.1) and finding a strong king in a tournament (Subsection 4.2) both admit a (randomized) query complexity of $\Theta(n^2)$. To do so, we make use of the constructions from Section 3.

4.1 Kings in Directed Graphs

Below is a property we require of Construction 8. Recall that a 2-element-set of vertices $\{v, w\}$ is said to be a *dominating pair* in $T = (V, E)$ if, for every $u \in V \setminus \{v, w\}$, there exists $x \in \{v, w\}$ such that $x \to u$.

▶ **Lemma 13.** *Let $n > 3$ be an odd positive integer, and let $i, j \in [n]_0$ with $i < j$. The set of vertices $\{i, j\}$ is a dominating pair in Δ_n if and only if $j = n-1$.*

Proof. We show both directions below.

- Let us first show the implication ($\Rightarrow$): that is, any pair $\{i, j\}$ where $j = n-1$ is dominating. According to Construction 8, and as depicted in Figure 1, vertex $n-1$ dominates all vertices except $n-2$. However, it is easy to see that all other vertices are either $n-2$ or dominate $n-2$. Therefore, pairing $n-1$ with any of those vertices produces a dominating pair.
- To show the implication in the other direction ($\Leftarrow$), assume, for the sake of contradiction, that $j \neq n-1$ and $\{i, j\}$ is a dominating pair. Then, $i \in [n-2]_0$ and either $j \in [n-2]_0$, or $j = n-2$ (recall that $i < j$).
  - If $j \in [n-2]_0$, then $\{i, j\}$ does not dominate $n-1$, a contradiction.
  - If $j = n-2$, then $i \in [n-2]_0$ must dominate all other vertices in $[n-2]_0$, and thus must be a source in $\Delta_n[[n-2]_0]$. However, since all (of the at least 3) vertices in $\Delta_n[[n-2]_0]$ are kings by Lemma 9, this contradicts Lemma 5. Thus, the pair $\{i, j\}$ is not a dominating pair.

This concludes the proof. ◀

Simply, the only dominating pairs in Δ_n are $\{0, n-1\}, \{1, n-1\}, \ldots, \{n-2, n-1\}$. Using the above property, we show below that the deterministic and randomized query complexity of finding a king in an arbitrary n-vertex digraph is $\Theta(n^2)$.

▶ **Theorem 14.** *For all positive integers n, $\mathsf{R}(\textsf{EXIST-KING}_n) = \Theta(n^2)$ for general digraphs.*

Proof. The upper bound is trivial. Assume that n is odd. If n is even, an argument similar to that given in the last paragraph of the proof of Theorem 18 can be used. Let $A = ([n]_0, E_A)$ be an arbitrary balanced tournament on n vertices. Consider the n-vertex tournament Δ_n from Construction 8 and relabel all vertex labels by adding n to them. Let the resultant graph be called A'. Construct a tournament C on $2n$ vertices labelled in $[2n]_0$ as follows:

1. Consider the disjoint union of A and A', and
2. add edges from every vertex in A to vertex $(2n-1)$ in A'.

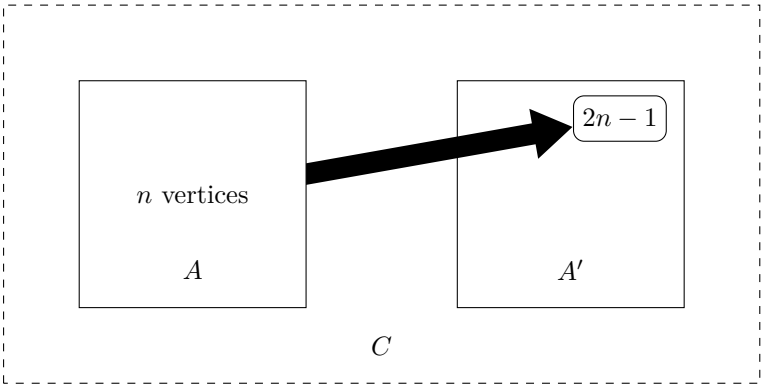


Figure 3 Construction of C from the proof of Theorem 14. $C[A]$ is a balanced tournament, and $C[A']$ is a relabelling of Δ_n from Construction 8.

Figure 3 illustrates the construction of C. Clearly, C does not have a king: indeed, no vertex in A' reaches any vertex in A, and every vertex in A dominates $(2n-1)$, which does not dominate $(2n-2)$ (by construction, see Construction 8).

For $i \in [n]_0$ and $j \in [n-1]_0$, define $C_{i,j}$ as the tournament obtained by adding the edge $i \to (n+j)$ in C. Observe that i is a king in $C_{i,j}$, since:

- i 2-step dominates all vertices in A (as $C_{i,j}[A]$ is a balanced tournament), and
- i dominates $(2n-1)$ and $n+j$, which form a dominating pair in $C_{i,j}[A']$ by Lemma 13.

Define the distribution μ on $2n$-vertex tournaments as follows:

$$\mu(T) = \begin{cases} 1/2 & T = C, \\ \frac{1}{2n(n-1)} & T = C_{i,j}, \quad \text{for all } i \in [n]_0, j \in [n-1]_0. \end{cases}$$

Equivalently, μ can be defined by the following sampling procedure: Start with the tournament C; then, with probability $1/2$, do nothing, or, with probability $1/2$, add a uniformly random (over $i \in [n]_0$ and $j \in [n-1]_0$) edge $i \to n+j$. Towards a contradiction using Lemma 7, let $\mathcal{A}$ be a deterministic algorithm with cost less than $n^2/100$ that decides whether a king exists in digraphs w.r.t. μ of error $\leq 1/3$.

Consider the computation path taken by $\mathcal{A}$ that answers all edges consistent with C, and answers 0 (i.e., "no edge present") to all queries of the form $i \to j$ for $i \in [n]$, $j \in \{n, n+1, \ldots, 2n-1\}$. We consider two cases.

1. Suppose the output at this leaf of $\mathcal{A}$ is 1 (i.e., "a king exists"). The algorithm errs on the input C, which has a μ-mass of 1/2. This is not possible since we assumed $\mathcal{A}$ to make an error of at most 1/3 w.r.t. μ.
2. Suppose the output at this leaf of $\mathcal{A}$ is 0 (i.e., "no king exists"). Since the number of queries on this path was assumed to be at most $n^2/100$, there must be at least $99n^2/100-n$ edges of the form $i \to (n+j)$ that have not been queried, where $i \in [n]_0$ and $j \in [n-1]_0$. For each such i, j, the graph $C_{i,j}$ also follows this computation path. However, the vertex i is a king in $C_{i,j}$. Hence $\mathcal{A}$ errs on $C_{i,j}$. Thus, the total error made by $\mathcal{A}$ is at least $(\frac{99n^2}{100} - n) \cdot \frac{1}{2n(n-1)} > 1/3$ for sufficiently large n. This again contradicts our assumption that $\mathcal{A}$ made error at most 1/3 w.r.t. μ, and hence the theorem follows.

This concludes the proof. ◀

As an easy corollary, we obtain the same deterministic lower bound for determining the existence of a king in a digraph.

▶ **Corollary 15.** *For all positive integers n, $\mathsf{D}(\text{EXIST-KING}_n) = \Theta(n^2)$ for general digraphs.*

4.2 Strong Kings in Tournaments

Recall that $\delta(v, u)$ denotes the number of length-2 paths from v to u in an underlying digraph/tournament. Recall that a strong king in a tournament is a king v such that, for every vertex u that dominates v (i.e., $u \to v$), we have $\delta(v, u) > \delta(u, v)$. Below, we show that in the construction U_n from Construction 10, for every vertex $v \in V(U_n)$ and an in-neighbor u of it, we have $\delta(v, u) - \delta(u, v) = 1$. In other words, there is exactly one more length-2 path from v to u than from u to v. This observation is crucial to argue that flipping the direction of a single edge used by v to reach u via a length-2 path will result in v no longer being a strong king.

▶ **Lemma 16.** *Let n be an odd positive integer. For each $j \to i$ in U_n, we have $\delta(i, j) - \delta(j, i) = 1$.*

Proof. Define the parity function $p(x) = x \pmod 2$. From the proof of Lemma 11, each vertex i in U_n dominates all lesser vertices of the same parity as i, and all greater vertices of opposite parity.

- Assume that $j > i$. Since $j \to i$, Construction 10 implies $p(j) = p(i)$. The vertex j dominates all vertices $v < j$ with $p(v) = p(i)$ and all vertices $w > j$ with $p(w) \neq p(i)$.
  To count length-2 paths from i to j, we first show that if $i \to v \to j$, then $i < v < j$.
  - If $v < i$, then $p(v) = p(i)$ (since $i \to v$), but also $p(v) \neq p(j)$ (since $v \to j$ and $v < j$), contradicting $p(i) = p(j)$.
  - If $v > j$, then $p(v) \neq p(i)$ (since $i \to v$), but also $p(v) = p(j)$ (since $v \to j$ and $v > j$); again, a contradiction.

  Thus, v must satisfy $i < v < j$, and since $i \to v$, it follows that $p(v) \neq p(i)$. This means $\delta(i, j) = |\{v : i < v < j \text{ and } p(v) \neq p(i)\}| = \frac{j-i}{2}$.
  Using nearly the same argument, one can show that $\delta(j, i) = |\{v : i < v < j \text{ and } p(v) = p(i)\}| = \frac{j-i}{2} - 1$.
- The case where $j < i$ follows from the case where $j > i$ by symmetry (Lemma 12). Indeed, let i' and j' be the new labels of i and j in $\sigma(U_n, n-i)$, respectively. Then, $i' = 0 < j'$, and given that $\sigma(U_n, n-i) \in \mathrm{Aut}(U_n)$, the lemma follows.

Therefore, we have $\delta(i, j) - \delta(j, i) = \frac{j-i}{2} - (\frac{j-i}{2} - 1) = 1$. ◀

For the tournament U_n and a vertex v of it, define $D(v)$ be the set of edges such that, if the direction of exactly one edge in $D(v)$ were flipped, v would no longer be a strong king.

Now, we show that $|D(v)| = \Theta(n^2)$.

▶ **Lemma 17.** *Let n be an odd positive integer. For the tournament U_n and $D(\cdot)$ as defined above, we have $|D(v)| = (n^2-1)/4$ for all $v \in [n]$.*

Proof. We show below that $|D(0)| = (n^2-1)/4$. The result for general v immediately follows by symmetry (Lemma 12), as $|D(0)| = |D(1)| = \cdots = |D(n-1)|$.

We first consider the effect of flipping an edge connected to 0.

- Let $j \in [n]_0$ be odd. The construction of U_n ensures the direction of the edge between 0 and j to be $0 \to j$. Flipping $(0, j)$ decreases $\delta(0, n-1)$ by 1 as the path $0 \to j \to n-1$ is gone, and does not affect $\delta(n-1, 0)$.
- Let $k \in [n]_0 \setminus \{0\}$ be even. The construction of U_n ensures the direction of the edge between 0 and k to be $k \to 0$. Flipping the $(k, 0)$ edge cannot increase $\delta(j, 0)$ for any j, and cannot decrease $\delta(0, j')$ for any j'. Since 0 was a strong king in U_n, this means 0 is a strong king in the new tournament with $(k, 0)$ flipped as well.

The first case above contributes $(n-1)/2$ elements to $D(0)$. Next, we consider the effect of flipping an edge between j and k where $j \neq 0$ and $k \neq 0$.

- If j and k are both odd, then neither of them is an in-neighbor of 0, and hence 0 remains a strong king.
- If j and k are both even and $j \to k$ is flipped, the construction of U_n ensures that $j > k$, and both j, k are in-neighbors of 0. We have $\delta(k, 0)$ increase by 1 due to the new path $k \to j \to 0$, and $\delta(0, k)$ is unchanged as $j \to 0$. By Lemma 16, the new tournament now has $\delta(k, 0) = \delta(0, k)$, and hence 0 is no longer a strong king.
- If j is even and $k > j$ is odd, the direction of the edge between j and k is $j \to k$. The vertex j is an in-neighbor of 0, and k is not. Flipping the edge increases $\delta(0, j)$ by 1 due to the new path $0 \to k \to j$, and $\delta(j, 0)$ is unaffected. No other values of $\delta(0, \cdot)$ or $\delta(\cdot, 0)$ are affected for in-neighbors of 0. Thus, 0 remains a strong king in this case.
- The final case is when j is odd and $k > j$ is even. Recall that k is an in-neighbor of 0. We have $\delta(0, k)$ *reduce* by 1 due to the removal of the path $0 \to j \to k$ (as the $j \to k$ edge is now flipped), and $\delta(k, 0)$ is unaffected. By Lemma 16, we have $\delta(0, k) = \delta(k, 0)$ in the new tournament, and hence 0 is no longer a strong king.

The number of edges from the second case above equals the number of ways of choosing two even numbers, none of which is 0: this is $\binom{(n-1)/2}{2}$. The number of edges from the fourth case is the number of choices of an odd vertex and a higher even vertex, which equals $(n-1)/2 + (n-3)/2 + \cdots + 1 = (n^2-1)/8$. Combining the above, we have

$$\begin{aligned} |D(0)| &= \frac{n-1}{2} + \frac{(n-1)(n-3)}{8} + \frac{n^2-1}{8} = \frac{n-1}{2}\left[1 + \frac{n-3}{4} + \frac{n+1}{4}\right] \\ &= \left(\frac{n-1}{2}\right)\left(\frac{n+1}{2}\right) = \frac{n^2-1}{4}. \end{aligned}$$

This concludes the proof. ◀

We now analyze the randomized query complexity of $\mathsf{STRONG\text{-}KING}_n$.

▶ **Theorem 18.** *For all positive integers n, $\mathsf{R}(\mathsf{STRONG\text{-}KING}_n) = \Theta(n^2)$ for tournaments.*

Proof. The upper bound is trivial. Assume that n is odd. For $i, j \in [n]_0$, let $U_n^{i,j}$ denote the tournament that is obtained from U_n by flipping the direction of the edge between i and j. Consider μ to be the following distribution on n-vertex tournaments:

$$\mu(T) = \begin{cases} 1/2 & T = U_n, \\ \frac{1}{\binom{n}{2}} & T = U_n^{i,j}, \end{cases} \qquad \text{for all } i,j \in [n]_0.$$

Towards a contradiction using Lemma 7, let $\mathcal{A}$ be a deterministic algorithm with cost less than $c \cdot n(n-1)$ (for some constant $c \in (0,1)$ to be fixed later in the proof, it will turn out that any $c < 1/12$ works) computing $\mathsf{STRONG\text{-}KING}_n$ to error less than $1/3$ when inputs are drawn from the distribution μ. Consider the computation path of $\mathcal{A}$ that outputs the answers of all queries consistently with the input U_n. Say the output at this leaf of $\mathcal{A}$ is vertex v. The algorithm errs on all inputs of the form $U_n^{i,j}$ where $(i,j) \in D(v)$ (recall that $D(v)$ consists of precisely those edges that, on being flipped in U_n, cause v to no longer be a strong king). There are at least $|D(v)| - cn(n-1)$ such edges (i,j) which are unqueried (and thus $U_n^{i,j}$ reaches this leaf). The error probability of $\mathcal{A}$ is at least its error probability on this leaf, which by Lemma 17 is at least $\frac{1}{\binom{n}{2}}\left(\frac{n^2-1}{4} - cn(n-1)\right) > \frac{2}{n(n-1)}\left(\frac{n(n-1)}{4} - cn(n-1)\right) = \frac{1-4c}{2} > 1/3$ for $c < 1/12$. Lemma 7 yields the required lower bound.

The proof can be easily made to work for the case where n is even by assuming that one vertex is a sink, and applying the argument above on the subtournament induced on the remaining vertices (this sink vertex does not affect the proof above in any way). ◀

A deterministic lower bound follows as an immediate corollary.

▶ **Corollary 19.** *For all positive integers n, $\mathsf{D}(\mathsf{STRONG\text{-}KING}_n) = \Theta(n^2)$ for tournaments.*

5 Conclusion

In this paper, we have shown the deterministic and randomized query complexities of deciding whether a king exists in an arbitrary n-vertex digraph to be $\Theta(n^2)$. We also showed that deterministic and randomized query complexities of finding a strong king in a tournament is $\Theta(n^2)$, answering a question of Biswas et al. [2] in the negative. Our proofs relied on showing some key properties of known constructions of balanced tournaments (tournaments where every vertex is a king). The most interesting question that remains open is to determine the deterministic query complexity of finding a king in an n-vertex tournament. The best-known upper and lower bounds are $O(n^{3/2})$ and $\Omega(n^{4/3})$, respectively, and are from over 20 years ago [19]. Perhaps our analysis of special balanced tournaments in this paper can provide insight towards approaching this general problem.

References

1 Amir Abboud, Tomer Grossman, Moni Naor, and Tomer Solomon. From donkeys to kings in tournaments. In *32nd Annual European Symposium on Algorithms, ESA*, volume 308 of *LIPIcs*, pages 3:1–3:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.3.

2 Arindam Biswas, Varunkumar Jayapaul, Venkatesh Raman, and Srinivasa Rao Satti. Finding kings in tournaments. *Discret. Appl. Math.*, 322:240–252, 2022. doi:10.1016/j.dam.2022.08.014.

3 Amit Chakrabarti and Subhash Khot. Improved lower bounds on the randomized complexity of graph properties. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 285–296. Springer, 2001. doi:10.1007/3-540-48224-5_24.

4 An-Hang Chen, Jou-Ming Chang, Yuwen Cheng, and Yue-Li Wang. The existence and uniqueness of strong kings in tournaments. *Discrete mathematics*, 308(12):2629–2633, 2008. doi:10.1016/j.disc.2007.05.008.

5 Palash Dey. Query complexity of tournament solutions. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2992–2998. AAAI Press, 2017. doi:10.1609/aaai.v31i1.10702.

6 David C Fisher, J Richard Lundgren, Sarah K Merz, and K Brooks Reid. The domination and competition graphs of a tournament. *Journal of Graph Theory*, 29(2):103–110, 1998. doi:10.1002/(SICI)1097-0118(199810)29:2\%3C103::AID-JGT6\%3E3.0.CO;2-V.

7 Péter Hajnal. An $\Omega(n^{4/3})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11:131–143, 1991. doi:10.1007/BF01206357.

8 Ting-Yem Ho and Jou-Ming Chang. Sorting a sequence of strong kings in a tournament. *Information processing letters*, 87(6):317–320, 2003. doi:10.1016/S0020-0190(03)00346-6.

9 Oded Lachish, Felix Reidl, and Chhaya Trehan. When you come at the king you best not miss. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022*, volume 250 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSTTCS.2022.25.

10 HG Landau. On dominance relations and the structure of animal societies: III the condition for a score structure. *The bulletin of mathematical biophysics*, 15:143–148, 1953.

11 Nikhil S. Mande, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the communication complexity of finding a king in a tournament. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 317 of *LIPIcs*, pages 64:1–64:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.64.

12 Nikhil S. Mande, Manaswi Paraashar, and Nitin Saurabh. Randomized and quantum query complexities of finding a king in a tournament. In *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 284 of *LIPIcs*, pages 30:1–30:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.FSTTCS.2023.30.

13 Stephen B Maurer. The king chicken theorems. *Mathematics Magazine*, 53(2):67–80, 1980.

14 John W Moon. *Topics on tournaments in graph theory*. Courier Dover Publications, 2015.

15 Nathan Muyinda, Bernard De Baets, and Shodhan Rao. Non-king elimination, intransitive triad interactions, and species coexistence in ecological competition networks. *Theoretical Ecology*, 13(3):385–397, 2020.

16 K.B. Reid. Every vertex a king. *Discrete Mathematics*, 38(1):93–98, 1982. doi:10.1016/0012-365X(82)90173-X.

17 Ronald L Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3(3):371–384, 1976. doi:10.1016/0304-3975(76)90053-0.

18 Arnold L Rosenberg. On the time required to recognize properties of graphs: A problem. *ACM SIGACT News*, 5(4):15–16, 1973. doi:10.1145/1008299.1008302.

19 Jian Shen, Li Sheng, and Jie Wu. Searching for sorted sequences of kings in tournaments. *SIAM J. Comput.*, 32(5):1201–1209, 2003. doi:10.1137/S0097539702410053.

20 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 222–227. IEEE Computer Society, 1977.

21 Andrew Chi-Chih Yao. Lower bounds to randomized algorithms for graph properties. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 393–400. IEEE, 1987.

Communication Complexity of Equality and Error-Correcting Codes

Dale Jacobs
Tufts University, Medford, MA, USA

John Jeang
Tufts University, Medford, MA, USA

Vladimir Podolskii
Tufts University, Medford, MA, USA

Morgan Prior
Tufts University, Medford, MA, USA

Ilya Volkovich
Boston College, MA, USA

Abstract

We study the public-coin randomized communication complexity of the equality function. The communication complexity of this function is known to be low when the error probability is constant and the players have access to many random bits. The complexity grows, however, if the allowed error probability and the amount of randomness are restricted.

We show that public-coin randomized protocols for equality and error-correcting codes are essentially the same object. That is, given a protocol for equality, we can construct a code, and vice versa.

We substantially extend the protocol-implies-code direction: any protocol computing a function with a large fooling set can be converted into an error-correcting code. As a corollary, we show that among functions with a fooling set of size s, equality on $\log s$ bits has the least randomized communication complexity, regardless of the restrictions on the error probability and the amount of randomness.

Finally, we use the connection to error-correcting codes to analyze the randomized communication complexity of equality for varying restrictions on the error probability and the amount of randomness. In most cases, we provide tight bounds. We pinpoint the setting in which tight bounds are still unknown.

2012 ACM Subject Classification Theory of computation → Communication complexity; Theory of computation → Error-correcting codes; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases communication complexity, randomized communication complexity, error-correcting codes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.37

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2025/068/` [9]

1 Introduction

In the standard model of communication complexity, two players, Alice and Bob, are tasked with computing the value of a fixed function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ on some given input (x, y). However, Alice only knows x and Bob only knows y, so they must communicate to compute $f(x, y)$. Communication complexity studies the amount of communication between players in terms of the number of bits exchanged. In recent decades, communication


© Dale Jacobs, John Jeang, Vladimir Podolskii, Morgan Prior, and Ilya Volkovich;
licensed under Creative Commons License CC-BY 4.0



complexity has become one of the central areas in computational complexity, with numerous applications to other areas, including circuit complexity, proof complexity, online algorithms, and data structures [11, 19, 22, 10].

The equality function $\mathrm{EQ}\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is one of the most fundamental functions in communication complexity. On input (x, y), this function outputs 1 iff $x = y$. This was the first function for which an exponential separation between deterministic and randomized communication complexity was shown (see [23]). Later, EQ played a key role in many results in communication complexity, including direct sum results for the randomized setting [7], protocols for threshold functions [17], lifting theorems [14, 6], and communication complexity with an equality oracle [5, 18].

It is well known that the randomized communication complexity of EQ is low for constant error probability. We can consider both the private-coin and public-coin models, which we sometimes refer to as the private randomness setting and the public randomness setting, respectively. In the private randomness setting, there is an $O(\log n)$ protocol, usually attributed to Rabin and Yao (unpublished, see [11, 20]). For the case of public randomness, there is an $O(1)$ protocol. The general idea of this protocol is that players can first use their shared randomness to sample a random hash function h. Then Alice sends Bob $h(x)$, and Bob compares it to $h(y)$. The most standard protocol of this type is for players to pick a random vector $r \in \{0,1\}^n$ and compare the inner products $\langle x, r\rangle$ and $\langle y, r\rangle$ over $\mathbb{F}_2$. In the case that x and y are not equal, there is a $1/2$ probability that the compared inner products are also not equal.

Although the public-coin randomized complexity of EQ is $O(1)$ for constant error probability and unlimited access to randomness, the complexity can increase when we impose stronger restrictions on these parameters. The error probability for EQ (as well as for any other function) can be reduced to any ε at the cost of a $\log \frac{1}{\varepsilon}$ factor in complexity by a standard error reduction procedure. Note that the EQ protocol above uses a large amount of randomness, which grows as the error probability is reduced. The number of random bits, which we denote by m, can be substantially reduced using the idea from Newman's result on the connection between communication complexity in the public-coin and private-coin models [16, 13]. Another known construction of randomized protocols for EQ with restricted randomness is through error-correcting codes. In such a protocol, players privately encode their inputs via an error-correcting code and compare them in a random coordinate (see Section 2.4 or [19, Chapter 3]).

As for lower bounds, Canetti and Goldreich [4] obtained general lower bounds in the setting with restricted randomness (by a reduction to the deterministic case; see Fact 14). For EQ, their result gives a lower bound of $\Omega\left(\frac{n}{\varepsilon 2^m}\right)$. They also provided examples of matching upper bounds for the case of constant error probability. Moran et al. [15] (see also [13]) gave a lower bound on communication complexity in the private-coin model in the small error case for functions with large fooling sets. In particular, for EQ, their bound implies a lower bound of $\Omega\left(\log n + \log \frac{1}{\varepsilon}\right)$ in the private randomness setting. For public randomness, this implies a lower bound $\Omega\left(\log n + \log \frac{1}{\varepsilon} - m\right)$. (Note that to reduce a protocol with private randomness to one with public randomness, one of the players can generate random bits and send them to another player.) Despite these results, the entire landscape of communication complexity of EQ for varying ε and m is not fully clear.

Our Results

As mentioned above, error-correcting codes give communication protocols for EQ. In our first result, we observe that actually, the converse is also true: communication protocols for EQ give rise to error-correcting codes. Thus, error-correcting codes and protocols for EQ are essentially the same object.

▶ **Theorem 1** (Informal version of Theorem 30 and Corollary 31). *Given a protocol for* EQ*, we can construct an error-correcting code with equivalent parameters (up to a constant factor), and vice versa. Furthermore, this connection is precise for one-sided error protocols; in this case, protocols and error-correcting codes are in one-to-one correspondence.*

We obtain the following corollary:

▶ **Corollary 2** (Informal version of Corollary 32). *Any randomized public-coin protocol for* EQ *can be converted into a one-sided false-positive error protocol with no loss in parameters.*

In particular, any communication protocol for EQ with ε_1 false positive error and ε_0 false negative error can be converted into a one-sided error protocol with $\varepsilon_0 + \varepsilon_1$ false positive error (the communication complexity and the number of random bits used remain the same).

Next, we show that one direction of the result above generalizes to arbitrary functions in the following way:

▶ **Theorem 3** (Informal version of Theorem 33). *Any randomized public-coin protocol for a function with a large fooling set can be converted into an error-correcting code with the same parameters as for* EQ*.*

As a consequence, we obtain the following corollary:

▶ **Corollary 4** (Informal version of Corollary 34). *Among all functions with a fooling set of size s,* EQ *on $\log s$ bits has the least randomized public-coin communication complexity for all values of error probability and number of random bits.*

In other words, this corollary says that of all functions, EQ provides the largest separation between fooling set size and public-coin randomized communication complexity.

We next use the established connection between communication complexity of EQ and error-correcting codes to provide a detailed analysis of the communication complexity k of EQ on n-bit inputs for varying amounts of random bits m and error ε. We start by translating bounds for error-correcting codes to lower bounds for communication complexity. In particular, based on Plotkin bound for codes, we obtain the following lower bound on the randomized communication complexity of EQ in the public-coin model (up to a constant factor):

▶ **Theorem 5** (Informal version of Theorem 35). *Any randomized public-coin protocol for* EQ *satisfies the following lower bound for the communication complexity k:*

$$k \geq \max\left(\min\left(n, \frac{n}{\varepsilon 2^m}\right), \log\frac{1}{\varepsilon}\right).$$

Furthermore, this lower bound is tight for many regimes of the parameters m, ε.

It is noted in [1] that Plotkin bound is the best known for the codes over large alphabet. We confirm this comparing it with Singleton bound, Elias-Bassalygo bound, Griesmer bound and MRRW bound. We observe that actually the combination of Singleton bound and Elias-Bassalygo bound gives the full lower bound in the theorem above (up to a constant factor).

Interestingly, the two known lower bounds for EQ mentioned above have natural interpretations in terms of codes. The result of [15] corresponds to the Elias-Bassalygo bound (after translation of their result to the public randomness setting) and the result of [4] matches the Singleton bound.

Next, we further analyze the landscape of communication complexity of EQ. Depending on values of the parameters m and ε, the expression in the theorem above can resolve (up to constant factors) to n, $\frac{n}{\varepsilon 2^m}$, or $\log \frac{1}{\varepsilon}$. In the first case, the trivial deterministic protocol for EQ (see Section 2.2) is clearly optimal. In the second case, we show that for a wide range of parameters, the protocol for EQ obtained from Reed-Solomon codes is optimal. However, this code does not apply in a certain range of parameters, specifically when $\log \frac{1}{\varepsilon} < \frac{n}{\varepsilon 2^m} < m$. In this case, we devise a protocol that is suboptimal, but close to the lower bound (specifically, we iterate the Reed-Solomon code with itself). Finally, in the third case, we show that the standard protocol combined with an improvement of Newman's theorem (see Fact 15) for reducing the number of random bits gives an upper bound that is tight, up to additive lower order terms. Previously this was known only for the case of constant ε [4].

Other related work

Fleischer et al. [8] studied the relation between the number of random bits and the zero-error communication complexity for the list-non-disjointness function. Ball et al. [2] gave a lower bound on randomized communication complexity with defective randomness through the size of the fooling set. The paper [15] analyses communication protocols of functions by looking at the statistical distance of the protocol on distinct inputs from the fooling set. The difference with our approach is that we look at Hamming distance between the vectors of outcomes instead.

One could apply round elimination techniques (e.g., [3]) to transform an arbitrary protocol for EQ into a one-way protocol, thus obtaining an alternative proof of Corollary 2 (or Corollary 32). This allows for an alternative prove of the translation from protocols to codes in Theorem 1: we can prove this translation for one-way protocol and use round elimination to extend the result to general protocols. However, since the proof of this direction of the theorem does not differ that much for one-way and for general communication protocols, we decided to give a direct proof for the general case.

The rest of the paper is organized as follows. In Section 2, we give the necessary definitions and state known results. In Section 3, we prove our results on the connection between communication complexity and error-correcting codes. In Section 4, we use the connection established in Section 3 to deduce lower bounds on the communication complexity of equality for various amounts of allowed error and randomness (the analysis of suboptimal bounds on the codes is omitted and can be found in the full version of the paper [9]). In Section 5, we analyze the landscape of communication complexity of equality by comparing our lower bounds against upper bounds for various regimes of parameters.

2 Preliminaries

We make frequent use of the following fact:

▶ **Fact 6.** *For two non-negative functions $a(n), b(n)$ such that $a(n) = b(n) + \log b(n)$,*

$$b(n) = a(n) - \log a(n) + O(1).$$

2.1 Notation and Conventions

All logarithms are base 2 unless otherwise specified. We use the notation $f^{(i)}$ to denote the function f composed with itself i times.

For brevity, unless specified otherwise, when we refer to the "complexity" of a function, we are referring to the randomized communication complexity in the public-coin model. Similarly, when we refer to a "protocol", we mean a randomized communication protocol in the public-coin model. By "naive protocol" for EQ, we mean the deterministic protocol in which Alice sends her entire input, and by the "standard protocol" for EQ, we mean the standard randomized protocol used in the public-coin model (see Chapter 3 of [11]. Finally, when we refer to a "code", we are referring to an error-correcting code.

We use m to denote the amount (in bits) of public randomness in a communication protocol, and we use ε to denote the allowed error. When protocols have two-sided error, we use ε_1 to denote false positive error and ε_0 to denote false negative error of a protocol. That is, ε_1 is an upper bound on the probability that the protocol outputs 1 on an input where the function outputs 0. The false negative error, ε_0, is an upper bound on the probability that the protocol outputs 0 on an input where the function outputs 1. For convenience, we often use k to denote public-coin randomized communication complexity. That is, we let k denote CC^{pub}. We use the notation $[N, K, D]$ code to refer to a code with block length N, message length K, and distance D (see Section 2.3 for definitions of these parameters).

2.2 Communication Complexity

In the standard model of communication complexity, two players, Alice and Bob, wish to compute the value of a function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ on some given input (x, y). The challenge is that Alice only knows $x \in \{0,1\}^n$ and Bob only knows $y \in \{0,1\}^n$, and thus the players need to communicate (i.e., send bits) to compute $f(x, y)$.

▶ **Definition 7** (protocol, value, cost, one-way)**.** *A* communication protocol Π *is a binary tree where each internal node v is labeled either by a function $a_v : \{0,1\}^n \to \{0,1\}$ or by a function $b_v : \{0,1\}^n \to \{0,1\}$ (denoting Alice or Bob's message respectively) and each leaf node is labeled with an element in $\{0,1\}$.*

The value *of the protocol Π on input (x, y) is the label reached by traversing the tree according to the following rules: start at the root, and for each internal node v labeled by a_v (resp b_v), move left if $a_v(x) = 0$ (resp $b_v(y) = 0$) and right otherwise. The* cost *of Π on input (x, y) is the length of the path taken on input (x, y). The* cost *of Π is the height of the tree. We call a protocol* one-way *if the entire protocol consists of just one message.*

▶ **Definition 8** (deterministic communication complexity)**.** *The* deterministic communication complexity *$CC(f)$ of $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is the minimum cost of Π, over all protocols Π that compute f.*

▶ **Fact 9** (deterministic communication complexity of EQ_n)**.** *For $x, y \in \{0,1\}^n$, the equality function, $\text{EQ}_n(x, y)$, is defined to be 1 if $x = y$ and 0 otherwise. We have that $CC(\text{EQ}_n) = n + 1$.*

The lower bound on $CC(\text{EQ})$ can be proved using the fooling set method (see [11]). We define a fooling set below:

▶ **Definition 10** (fooling set [11])**.** *A fooling set for $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is a set $F \subseteq \{0,1\}^n \times \{0,1\}^n$ and a value $b \in \{0,1\}$ such that*

1. *For every $(x, y) \in F$, $f(x, y) = b$*
2. *For all distinct pairs $(x_1, y_1), (x_2, y_2)$, either $f(x_1, y_2) \neq b$ or $f(x_2, y_1) \neq b$.*

While Definition 7 defines protocols for deterministic communication complexity, this paper focuses on randomized communication complexity, wherein Alice and Bob have a source of public randomness. Formally, there is a random string of bits $r \in \{0,1\}^*$ available

to both players. Where in the deterministic setting we had internal nodes in the protocol tree labeled with $a_v(x)$ and $b_v(y)$, we now have internal nodes labeled with $a_v(x,r)$ and $b_v(y,r)$; that is, Alice and Bob's communication can depend on r. We can also view this model as a distribution $\{P_r\}_{r\in\Pi}$ on deterministic protocols:

▶ **Definition 11** (randomized public-coin protocol [11]). *A* (randomized) public-coin protocol *is a probability distribution over deterministic protocols. The* success probability *of a public-coin protocol on input (x,y) is the probability of choosing a (deterministic) protocol, according to the probability distribution $\{P_r\}_{r\in\Pi}$, that computes $f(x,y)$ correctly.*

▶ **Definition 12** (randomized communication complexity [11]). *The* randomized communication complexity $CC^{pub}_{\varepsilon}(f)$ *is the minimum cost of a public-coin protocol that computes f with an error of at most ε (for every input (x,y)).*

▶ **Fact 13** (randomized communication complexity of EQ). *The randomized communication complexity (in the public randomness model) of* EQ *with constant error ε is $O(1)$.*

A proof of Fact 13 can be found in [11].

Randomized protocols can be converted to deterministic protocols, with some overhead on the communication complexity, by repeating the randomized procedure and taking a majority vote. The following upper bound is proved in [4]:

▶ **Fact 14** ([4]). *Let $f\colon \{0,1\}^n\times\{0,1\}^n\to\{0,1\}$, be a function computable via a randomized communication protocol Π using k bits of communication, m bits of randomness, and with error probability ε. Then the deterministic communication complexity $CC(f)$ is bounded by*

$$CC(f)\le k(\varepsilon 2^{m+1}+1). \tag{1}$$

The following fact improves upon Newman's Theorem [16], and states that a public-coin protocol can be converted into a private-coin protocol with only logarithmic additive overhead in the complexity. Similarly, the amount of randomness used by a public-coin protocol can be reduced by incurring a small amount of extra error.

▶ **Fact 15** ([13]). *Let $f:\{0,1\}^n\times\{0,1\}^n\to\{0,1\}$ be a function. Then $CC^{priv}_{\varepsilon(1+\delta)}(f)\le CC^{pub}_{\varepsilon}(f)+\log(\frac{n}{\varepsilon})+\log(\frac{6}{\delta^2})$. Furthermore, any public-coin protocol for f which uses m random bits and has error ϵ can be converted into a protocol using $\log n+\log(1/\varepsilon)+\log(\frac{6}{\delta^2})$ random bits with error $\epsilon(1+\delta)$, at no penalty to the communication complexity.*

Interestingly, this improvement over original Newman's Theorem is crucial for tight bounds in one of the cases in our analysis in Section 5.3.

See [11] for more on private and public-coin settings in communication complexity.

2.3 Error-Correcting Codes and Bounds

▶ **Definition 16** (code, block length, codewords). *A* code C *of* block length N *over alphabet Σ is $C\subseteq\Sigma^n$. The elements $c\in C$ are called* codewords.

▶ **Definition 17** (distance, relative distance). *The* distance D *of a code $C\subseteq\Sigma^N$ is $min_{c\neq c'}(\Delta(c,c'))$, where Δ is the Hamming distance function. In other words, the* distance *of a code is the minimum Hamming distance between any pair of codewords. The* relative distance δ *of C is the ratio of distance to block length, $\frac{D}{N}$.*

▶ **Definition 18** (message length). *The* message length *(sometimes called* dimension*) K of a code C over alphabet Σ with $|\Sigma|=q$ is $K=\log_q|C|$.*

We call a code with block length N, message length K, and distance D an $[N, K, D]$ code.

▶ **Definition 19** (rate). *The* rate *of a code with block length N and message length K is* $R := \frac{K}{N}$.

High distance and high rate are desirable properties for codes; however, there is a trade-off between these two parameters. The bounds stated below offer some insight into this trade-off.

▶ **Fact 20** (Singleton Bound). *Let C be an $[N, K, D]$ code with alphabet size q. Then*

$$K \leq N - D + 1.$$

Equivalently, we can express the Singleton Bound in terms of rate and relative distance:

$$R \leq 1 - \delta + \frac{1}{N}.$$

The Singleton Bound is tight when $q \geq N$. That is, there exist codes for which $K = N - D + 1$ under this condition. We call such codes MDS (Maximum Distance Separable) codes. See Section 2.3 below.

▶ **Definition 21** (q-ary entropy). *The* q-ary entropy function $H_q(\delta)$, *a generalization of binary entropy, is defined as follows:*

$$H_q(\delta) := \delta \log_q \left(\frac{q-1}{\delta}\right) + (1-\delta) \log_q \left(\frac{1}{1-\delta}\right).$$

As the block length N of a code tends to infinity, we have the relationship $\mathrm{Vol}_q(\delta N, N) \approx q^{NH_q(\delta)}$, where $\mathrm{Vol}_q(\delta N, N)$ is the volume of the Hamming Ball with strings of length $\leq N$, alphabet size q, and radius δN. Hence H_q appears in the asymptotic statement of various bounds for error-correcting codes, such as the Elias-Bassalygo Bound below:

▶ **Fact 22** (Elias-Bassalygo Bound). *Let C be an $[N, K, D]$ code with alphabet size q. Then*

$$R := \frac{K}{N} \leq 1 - H_q\left[\left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - (\frac{q}{q-1})\delta}\right)\right].$$

▶ **Fact 23** (Griesmer Bound). *For a linear $[N, K, D]$ code with alphabet size q, we have:*

$$N \geq \sum_{i=0}^{K-1} \left\lceil \frac{D}{q^i} \right\rceil.$$

For more information on linear codes, see [21].

▶ **Fact 24** (Plotkin Bound). *The Plotkin bound strengthens the Singleton bound. For a code C with message length K, block length N, relative distance δ, and alphabet size q, we have:*

$$\frac{K}{N} + \delta\left(\frac{q}{q-1}\right) \leq 1 + \frac{1}{N}.$$

▶ **Fact 25** (MRRW Bound for large alphabet [1]). *Let C be an $[N, K, D]$ code with alphabet size q and $0 < \delta < (q-1)/q$. Then*

$$R := \frac{K}{N} \leq H_q\left[\left(q - 1 - (q-2)\delta - 2\sqrt{(q-1)\delta(1-\delta)}\right)/q\right].$$

Our upper bounds use protocols based on Reed-Solomon codes, which we define below. For more information about these codes, see [21].

Reed-Solomon codes are a family of codes which achieve the Singleton bound (i.e., for which $D = N - K + 1$).

Formally, Reed-Solomon codes are defined as follows:

▶ **Definition 26** (Reed-Solomon codes). *Let $q \geq N \geq K$. Let $\alpha_1, \ldots \alpha_N \in \mathbb{F}_q$ be distinct. The Reed-Solomon code over $\mathbb{F}_q$ with evaluation points $\overrightarrow{\alpha} = (\alpha_1, \ldots, \alpha_N)$, is the $[N, K, N-K+1]$ code given by:*

$$RS_q(N, K-1) = \{(f(\alpha_1), f(\alpha_2) \ldots, f(\alpha_N)) : f \in \mathbb{F}_q[X],\ deg(f) \leq K-1\}.$$

2.4 Codes Give Communication Protocols for EQ

It is known (see [19, beginning of Chapter 3]) that codes give communication protocols for EQ. In particular, we have the following fact:

▶ **Fact 27.** *For every $[N, K, D]$ code with alphabet Σ of size q, there exists a communication protocol, Π, computing EQ_n, such that $n \leq K \log q$, using m public random bits, and $\log q$ bits of communication, with error $\frac{N-D}{2^m}$. Moreover, this protocol is one-way and has one-sided false-positive error.*

3 Communication Complexity vs. Error-Correcting Codes

In this section, we show that randomized communication protocols for EQ give rise to error-correcting codes. In fact, we prove a generalization of this statement which applies to functions that are close to EQ in some sense, which we call "EQ-like" functions.

▶ **Definition 28** (EQ-like). *We say that a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is* EQ*-like if for all $x \in \{0,1\}^n$, $f(x,x) = 1$ and for all pairs (x,y) such that $x \neq y$, either $f(x,y) = 0$ or $f(y,x) = 0$.*

Note that EQ is a particular example of an EQ-like function, where for all pairs (x,y) such that $x \neq y$, $f(x,y) = 0$ ***and*** $f(y,x) = 0$. Another example of an EQ-like function is the well-known *Greater-Than* function, defined as follows: $\mathrm{GT}(x,y) = \mathbb{1}[int(x) \geq int(y)]$ where $int(x)$ is the integer given by x as its binary representation.

3.1 Protocols for EQ Imply Codes

Consider a protocol for f. Alice holds x, Bob holds y, and there is public randomness r. Let $T(x,y,r)$ be the transcript of Alice and Bob's communication. That is,

$$T(x,y,r) = (a_1, b_1, a_2, b_2, \ldots, a_R, b_R),$$

where each a_i is a function of all previous messages, x, and their shared randomness r, and each b_i is a function of all previous messages, y, and r. Without loss of generality, at the end of the protocol, Alice outputs $c(T(x,y,r), x, r)$. Let ε_1 and ε_0 denote the false positive and false negative error of the protocol, respectively. That is,

$$\forall (x,y) \text{ such that } f(x,y) = 0, \quad \Pr_r[c(T(x,y,r), x, r) = 1] \leq \varepsilon_1$$

$$\forall (x,y) \text{ such that } f(x,y) = 1, \quad \Pr_r[c(T(x,y,r), x, r) = 1] \geq 1 - \varepsilon_0.$$

We show that the transcript T of Alice and Bob's messages forms a code. An encoding of $x \in \{0,1\}^n$ is simply the concatenation of $T(x,x,r)$ for all choices of r.

▶ **Lemma 29.** *Let f be* EQ*-like, and let $T(x, y, r)$ be the transcript for Alice and Bob's communication in a randomized protocol for f as given above. Then for $x \neq y \in \{0,1\}^n$:*

$$\Pr_r[T(x,x,r) = T(y,y,r)] \leq \varepsilon_0 + \varepsilon_1.$$

Proof. Let $x \neq y \in \{0,1\}^n$. Without loss of generality suppose $f(x,y) = 0$ [1]. Then we have

$$\begin{aligned}
&\Pr_r[c(T(x,y,r),x,r) = 1] + \Pr_r[c(T(y,y,r),y,r) = 0] \geq \\
&\geq \Big(\Pr_r[c(T(x,y,r),x,r) = 1 \mid T(x,x,r) = T(y,y,r)] + \Pr_r[c(T(y,y,r),y,r) = 0 \mid T(x,x,r) = T(y,y,r)]\Big) \\
&\quad \cdot \Pr_r[T(x,x,r) = T(y,y,r)] = \\
&\Big(\Pr_r[c(T(y,y,r),y,r) = 1 \mid T(x,x,r) = T(y,y,r)] + \Pr_r[c(T(y,y,r),y,r) = 0 \mid T(x,x,r) = T(y,y,r)]\Big) \\
&\quad \cdot \Pr_r[T(x,x,r) = T(y,y,r)] = \\
&= 1 \cdot \Pr_r[T(x,x,r) = T(y,y,r)],
\end{aligned}$$

where the substitution $T(x,y,r) = T(y,y,r)$ follows from the fact that $T(x,x,r) = T(y,y,r)$ and that the communication tree partitions the inputs into combinatorial rectangles. Hence,

$$\Pr_r[T(x,x,r) = T(y,y,r)] \leq \Pr_r[c(T(x,y,r),x,r) = 1] + \Pr_r[c(T(y,y,r),y,r) = 0] \leq \varepsilon_0 + \varepsilon_1. \blacktriangleleft$$

Using Lemma 29, we obtain the following theorem:

▶ **Theorem 30.** *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be* EQ*-like, and suppose there exists a randomized communication protocol for f which has communication complexity k, uses m bits of public randomness, and has two-sided error probabilities ε_0 and ε_1. Then there exists an error-correcting code with message length n, codeword length 2^m, relative distance $1 - (\varepsilon_0 + \varepsilon_1)$, and alphabet size 2^k.*

Proof. The proof follows directly from Lemma 29. Concretely, the codeword for $x \in \{0,1\}^n$ is given by the concatenation of $T(x,x,r)$ for all 2^m possible values of randomness $r \in \{0,1\}^m$. Each symbol in the codeword corresponds to $T(x,x,r)$ for one choice of r, and since each transcript uses k bits of communication, there are 2^k possible symbols in the code alphabet. Finally, by Lemma 29, the probability that two different codewords agree on any given coordinate is bounded above by $\varepsilon_0 + \varepsilon_1$. Thus the relative distance of the code is $1 - (\varepsilon_0 + \varepsilon_1)$. ◀

If we consider a protocol with one-sided error, that is, a protocol with either $\varepsilon_0 = 0$ or $\varepsilon_1 = 0$, we immediately obtain the following corollary:

▶ **Corollary 31** (One-sided error)**.** *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be* EQ*-like, and suppose there exists a randomized communication protocol for f which has communication complexity k, uses m bits of public randomness, and has one-sided error probability ε. Then there exists a code with message length n, codeword length 2^m, relative distance $1 - \varepsilon$, and alphabet size 2^k.*

We remark that since codes give protocols (Fact 27), this corollary shows that one-way protocols for EQ with one-sided false-positive error are in one-to-one correspondence with error-correcting codes. That is, we can translate from protocols to codes and vice versa with no loss in parameters. Another consequence of Corollary 31 is the following corollary:

[1] If instead $f(y,x) = 0$, we replace $T(x,y,r)$ with $T(y,x,r)$ in the first two lines of Equation and the proof holds.

▶ **Corollary 32.** *Any randomized protocol for* EQ *can be converted to a one-way protocol with one-sided false-positive error with no loss in parameters.*

The following table summarizes the relationship between the communication complexity of a protocol for EQ and the obtained code:

Table 1 Translation Between CC and Codes.

Communication Complexity	Code
size of input $= n$	# of messages $= 2^n$
# of random bits $= m$	length of code $= 2^m$
error $= \varepsilon$	relative distance $= 1 - \varepsilon$
complexity $= k$	alphabet size $= 2^k$

3.2 Codes from Functions Based on Fooling Sets

We have seen that protocols for EQ-like functions give codes. We can actually make a more general statement which applies to all functions. In particular, for any function f, a protocol for f gives a code whose parameters depend on the parameters of the protocol and the size of the largest fooling set for f.

▶ **Theorem 33.** *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be any function with fooling set F, and suppose there exists a randomized communication protocol for f which has communication complexity k, uses m bits of randomness, and has error probabilities $\varepsilon_0, \varepsilon_1$. Then there exists a code with block length 2^m, message length $\log |F|$, relative distance $1 - (\varepsilon_0 + \varepsilon_1)$, and alphabet size 2^k.*

Proof. To prove the theorem, we show how to construct an error-correcting code from the function f with fooling set F. Let X and Y denote the set of x and y inputs respectively which appear in elements of the fooling set F, and define $f' : \{0,1\}^{\log|F|} \times \{0,1\}^{\log|F|} \to \{0,1\}$ to be f restricted to the inputs $X \times Y$. Without loss of generality, assume that for $(x, y) \in F$, $f(x, y) = 1$. By the definition of fooling set (see Section 2.2), f' is EQ-like, up to a permutation on one half of the input. Therefore, a protocol for computing f gives a protocol for f', which by Lemma 29 gives an error-correcting code for messages of length $\log |F|$. ◀

As a consequence of Theorem 33, we obtain the following corollary, which says that in some sense, EQ represents the largest gap between fooling set size and randomized communication complexity in the public-coin model.

▶ **Corollary 34.** *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be any function with fooling set F, and suppose there exists a randomized communication protocol for f which has communication complexity k, uses m bits of randomness, and has error probabilities $\varepsilon_0, \varepsilon_1$. Then there exists a protocol for* EQ *on $\log |F|$ bits which has communication complexity k, uses m bits of randomness, and has one-sided false-positive error probability $\varepsilon_0 + \varepsilon_1$.*

This follows because, given a protocol for f, we can construct a code with message length $\log |F|$, from which we obtain a protocol for EQ on $\log |F|$ bits.

Informally, Corollary 34 says that among all functions with a fooling set of size s, EQ on $\log s$ bits has the smallest public-coin randomized complexity for all values of ε and m. For protocols with two-sided error, this interpretation is true up to constant factors. For protocols with one-sided error, it is precise.

4 Communication Lower Bounds Implied by Code Bounds

The equivalence between EQ communication protocol and codes allows us to view bounds on codes as bounds on communication complexity. In particular, since we showed that existence of protocols implies existence of codes, nonexistence of codes implies nonexistence of protocols. Using this connection, we can translate known bounds for codes (Section 2.3) into bounds on communication complexity using the correspondence given by Table 1[2]. Below we describe the bound that can be obtained from Plotkin bound, that turns out to be the best known bound for large size of the alphabet.

The Plotkin bound (Fact 24) can be expressed in terms of communication protocols as:

$$\frac{n}{k2^m} + (1-\varepsilon)\left(\frac{2^k}{2^k-1}\right) \leq 1 + \frac{1}{2^m}. \tag{2}$$

In terms of error rate ε, this implies

$$\varepsilon \geq \underbrace{\frac{n-k}{2^m k}\left(\frac{2^k-1}{2^k}\right)}_{A} + \underbrace{\frac{1}{2^k}}_{B}. \tag{3}$$

From this we get the system

$$\begin{cases} \varepsilon \geq \frac{n-k}{2^m k}\left(\frac{2^k-1}{2^k}\right), \\ \varepsilon \geq \frac{1}{2^k}. \end{cases}$$

For any k we have $\left(\frac{2^k-1}{2^k}\right) \geq \frac{1}{2}$, thus solving for k we get the following (up to constant factors)

$$\begin{cases} k \geq \frac{n}{2^m\varepsilon+1}, \\ k \geq \log\frac{1}{\varepsilon}. \end{cases}$$

Depending on m and ε, either 1 or $\varepsilon 2^m$ is the dominant term in the denominator of the first inequality. Based on these two cases, this bound can be equivalently stated (up to constant factors) as $k \geq \min(\frac{n}{\varepsilon 2^m}, n)$.

Combining two bounds together we get the following theorem.

▶ **Theorem 35.** *For any randomized public-coin protocol for* EQ *on n bits with complexity k, m bits of randomness, and error probability ε,*

$$k \geq \max\left(\min(n, \frac{n}{\varepsilon 2^m}), \log\frac{1}{\varepsilon}\right) \tag{4}$$

up to constant factors.

It was noted (without details) in [1] that Plotkin bound is the best known bound for the case of error-correcting codes with large alphabet. For the sake of completeness, we provide a detailed analysis of communication complexity bounds we can obtain from Singleton bound, Elias-Bassalygo bound, Griesmer bound and MRRW bound. The analysis for these bounds

[2] Note that these lower bounds also hold for functions with large fooling sets.

can be found in the full version of our paper [9]. We also note that the combination of Singleton bound, Elias-Bassalygo bound matches Plotkin bound up to the constant factor (see [9]).

Note that Theorem 35 gives a different lower bound (one of n, $\frac{n}{\varepsilon 2^m}$, or $\log \frac{1}{\varepsilon}$) for different regimes of the parameters n, m, and ε. In Section 5, we examine the necessary relationship between n, m, and ε for achieving each of the three possible lower bounds. We also analyze when these bounds are tight by giving matching upper bounds for most regimes of the parameters. This analysis is summarized in the table below.

Table 2 Bounds on complexity of EQ for various regimes of parameters.

Description of the Regime	Lower Bound	Upper Bound
$\frac{1}{2^m} \geq \varepsilon \geq \frac{1}{2^n}$	n	n
$n \geq \frac{n}{\varepsilon 2^m} \geq \log \frac{1}{\varepsilon}$	$\frac{n}{\varepsilon 2^m}$	$\frac{n}{\varepsilon 2^m}$, if $m < \frac{n}{\varepsilon 2^m}$
		$\frac{n}{2^m}$, if $\log n - \log^{(c)} n > m > \frac{n}{\varepsilon 2^m}$ and constant ε and c
		$\frac{n}{\varepsilon^c 2^m}$, if $\varepsilon = \frac{1}{\log^{(c)} n}$ and c is constant
$\log n + \log \frac{1}{\varepsilon} - \log\log \frac{1}{\varepsilon} < m$	$\log \frac{1}{\varepsilon}$	$\log \frac{1}{\varepsilon}$, if $m > \log n + \log \frac{1}{\varepsilon} + \Theta(1)$

5 Analyzing the Landscape of Communication Complexity of EQ

In this section, we analyze the various regimes of parameters which give rise to different lower bounds on communication complexity. We further examine when these bounds are tight by giving upper bounds. Now we consider the three cases based on which of the three terms the maximum in Theorem 35 evaluates to.

5.1 Case 1

In this case, we have

$$\begin{cases} n \leq \frac{n}{\varepsilon 2^m}, \\ n \geq \log \frac{1}{\varepsilon}, \end{cases}$$

or, after rearranging, $\frac{1}{2^n} \leq \varepsilon \leq \frac{1}{2^m}$. In this case, $\max\left(\min(n, \frac{n}{\varepsilon 2^m}), \log \frac{1}{\varepsilon}\right) = n$, giving the lower bound $k \geq n$. Clearly, the naive protocol for EQ gives us the matching upper bound.

▶ Remark 36. This lower bound is actually easy to see, since for $\varepsilon < \frac{1}{2^m}$ only the deterministic protocol is possible (because of the granularity of error probability).

5.2 Case 2

In this case, we have

$$\begin{cases} n \geq \frac{n}{\varepsilon 2^m}, \\ \frac{n}{\varepsilon 2^m} \geq \log \frac{1}{\varepsilon}. \end{cases}$$

The lower bound in this case is $k \geq \frac{n}{\varepsilon 2^m}$.

▶ Remark 37. Note that this case shows that we cannot study the value of m up to a constant factor. The reason is that the bound depends on 2^m and multiplicative changes for m may affect the bound dramatically. Note that we never allowed extra multiplicative constant factors for m.

5.2.1 Upper Bound with Reed-Solomon Protocol

For the upper bound in this case, we can use the communication protocol derived from Reed-Solomon codes (Section 2.3). In this protocol, Alice and Bob encode their inputs using a Reed-Solomon code and check the value of a single random index in a subset S of their respective codewords. For some values of parameters, the communication complexity of this protocol matches the lower bound given by the Singleton bound.

▶ **Lemma 38.** *For any $m \leq k$, there exists a public-coin randomized communication protocol that computes* EQ *using k bits of communication, m bits of randomness, and error rate $\varepsilon = O(\frac{n}{k2^m})$.*

Proof. Let q be the largest prime number such that $q < 2^k$. To compute EQ, we use a $\mathrm{RS}_q(q, \lceil \frac{n}{k} \rceil)$ code to get an error rate of

$$\varepsilon = \frac{\lceil \frac{n}{k} \rceil}{2^m} = \frac{n}{k2^m} + \Theta\left(\frac{1}{2^m}\right), \tag{5}$$

but we know that $k \leq n + 1$ (the naive protocol for EQ gives this upper bound), giving $\varepsilon = O(\frac{n}{k2^m})$. ◀

Note that Reed-Solomon codes are only defined for $m \leq k$. For this protocol, $k = \Theta(\frac{n}{\varepsilon 2^m})$. Thus, we have a protocol only in the regime

$$m \leq \Theta(\frac{n}{\varepsilon 2^m}).$$

Hence, this protocol does not give an upper bound when

$$\log \frac{1}{\varepsilon} \leq \frac{n}{\varepsilon 2^m} \leq m. \tag{6}$$

An example parameters in this regime is $m = \log n$ and $\varepsilon = 1/\log\log n$. The lower bound in this case is $\frac{n}{\varepsilon 2^m} = \log\log n$. We partially address this issue in the next section.

5.2.2 Upper bound with iterated Reed-Solomon

The Reed-Solomon code allows us to do the following transformation of parameters: $n \longmapsto (m, \varepsilon, \frac{n}{\varepsilon 2^m})$. This notation means that when we need to send a message of length n, we can use m random bits, introduce an error ε, and reduce the problem to sending a message of size $\frac{n}{\varepsilon 2^m}$. This protocol gives a tight upper bound, but it has the restriction $m \leq \frac{n}{\varepsilon 2^m}$. For the range of parameters

$$\log \frac{1}{\varepsilon} < \frac{n}{\varepsilon 2^m} < m,$$

we do not have a tight upper bound.

Restating in terms of m and using Fact 6, Reed-Solomon codes give tight upper bounds for the regime of parameters satisfying

$$m \leq \log n + \log \frac{1}{\varepsilon} - \log\left(\log n + \log \frac{1}{\varepsilon}\right). \tag{7}$$

Thus, the following range of the lower bound is not matched with a tight upper bound:

$$\log n + \log \frac{1}{\varepsilon} - \log\left(\log n + \log \frac{1}{\varepsilon}\right) < m < \log n + \log \frac{1}{\varepsilon} - \log\log \frac{1}{\varepsilon}. \tag{8}$$

Our plan is to apply the Reed-Solomon code iteratively. That is, in the first step, we apply the code with some parameters:

$$n \longmapsto (m_1, \varepsilon_1, \frac{n}{\varepsilon_1 2^{m_1}}),$$

but then instead of sending $\frac{n}{\varepsilon_1 2^{m_1}}$ bits, we apply the Reed-Solomon code again (with some new values of parameters).

We note that the idea to apply codes iteratively is well known in coding theory under the name of concatenated codes [21]. In particular, it was previously observed in communication complexity language that applying Reed-Solomon codes iteratively helps to improve multiplicative constant and lower order terms in communication complexity of the EQ function in the regime of constant error probability and unlimited randomness [12]. However, we need a more detailed analysis.

For a real nonnegative x, we introduce the function $f(x) = \log(\log \frac{1}{\varepsilon} + x)$. We now describe the values of parameters that we can cover with c iterations of Reed-Solomon protocol. The proof of the following lemma can be found in Section A.1.

▶ **Lemma 39.** *For any integer constant c we have that the communication protocol derived from c iterations of Reed-Solomon has complexity $\frac{n}{\varepsilon^c 2^m}$ and is valid for all*

$$m \leq \log n + c \log \frac{1}{\varepsilon} - f^{(c)}(\log n) \tag{9}$$

(up to an additive constant), where $f^{(c)}$ is the function f applied to itself c times.

For example, for constant ε the lower bound we get is still tight up to the constant factor. It is not hard to see in this case that the function $f^{(c)}(\log n)$ is equal to $\log^{(c+1)} n$ up to an additive constant, and thus the range of parameters for which we do not have an upper bound reduces to

$$\log n - \log^{(c+1)} n < m < \log n.$$

For non-constant ε, our upper bound is not tight. To analyze the interval in which it applies, observe that $f(x) = \log(\log \frac{1}{\varepsilon} + x)$ is equal to $\log\log \frac{1}{\varepsilon}$ (up to an additive constant 1) for $\log \frac{1}{\varepsilon} \geq x$ and is equal to $\log x$ (up to an additive constant 1) for $\log \frac{1}{\varepsilon} \leq x$.

For example, for $\varepsilon = \frac{1}{\log^{(t)} n}$, for constant t, we get that $\log \frac{1}{\varepsilon} = \log^{(t+1)} n$. After $t+1$ iterations we have that $f^{(t+1)} = \log\log \frac{1}{\varepsilon}$ (up to an additive constant 1). Substituting this into (9) we get that this upper bound covers the whole regime (8).

5.3 Case 3

This case applies when $\log \frac{1}{\varepsilon} \geq n$ or $\log \frac{1}{\varepsilon} \geq \frac{n}{\varepsilon 2^m}$. The lower bound in this case is $k \geq \log \frac{1}{\varepsilon}$.

Note that the first of the subcases here corresponds to $\varepsilon \leq \frac{1}{2^n}$, in which case the communication complexity bound is n. We know that this bound is tight.

The remaining case to consider is $\log \frac{1}{\varepsilon} \geq \frac{n}{\varepsilon 2^m}$ which, after the rearranging gives $2^m \geq n \frac{1}{\varepsilon \log \frac{1}{\varepsilon}}$. This is equivalent to

$$m \geq \log n + \log \frac{1}{\varepsilon} - \log \log \frac{1}{\varepsilon}.$$

As for the upper bound, we use the standard randomized protocol and apply Fact 15. The standard protocol gives us the parameters

$$(n \log \frac{1}{\varepsilon}, \varepsilon, \log \frac{1}{\varepsilon}).$$

Applying Fact 15 with error parameter δ gives

$$(\log n + \log \frac{1}{\varepsilon} + \log \left(\frac{6}{\delta^2} \right), \varepsilon(1+\delta), \log \frac{1}{\varepsilon}).$$

Setting $\delta = \Theta(1)$ yields

$$(\log n + \log \frac{1}{\varepsilon} + \Theta(1), \Theta(\varepsilon), \log \frac{1}{\varepsilon}),$$

and this bound achieves optimal m up to an additive factor in lower order terms, and optimal ε up to a constant multiplicative factor.

▶ Remark 40. We observe that this public-coin protocol implies a private-coin protocol, where the randomness is simply shared at the start of the protocol at the cost of $\log n + \log \frac{1}{\varepsilon} + \Theta(1)$ bits of communication. This resulting private-coin protocol has $\Theta(\varepsilon)$ error using $\log(\frac{n}{\varepsilon^2}) + \Theta(1)$ bits of communication, essentially matching the lower bound of $\log(\frac{n}{\varepsilon^2}) - \log\log(\frac{1}{\varepsilon})$ observed in [13]. This indicates that in this range of parameters, the optimal private-coin protocol is to simply share the private randomness and then simulate the standard public-coin protocol.

6 Discussion & Future Directions

We identify an equivalence between randomized public-coin protocols for equality and error-correcting codes, and generalize one direction of this equivalence to functions with large fooling sets. We use this connection to study the randomized communication complexity of EQ for various regimes of the error ε and randomness m parameters. We give lower bounds for protocols from lower bounds from codes, and analyze when these lower bounds are tight. An immediate question is whether we can tighten our bounds in the regime in which they are not tight.

One way to view our result on fooling sets is that the maximal separation between the fooling set size and randomized communication complexity with public randomness is achieved by the equality function (for all regimes of parameters). A natural future direction is to explore whether similar separations can be found between randomized communication complexity and other complexity measures, such as deterministic communication complexity, rectangle size, or rank. Partial progress in this direction was obtained by Canetti and Goldreich [4].

References

1 Matti J. Aaltonen. Linear programming bounds for tree codes (corresp.). *IEEE Trans. Inf. Theory*, 25(1):85–90, 1979. doi:10.1109/TIT.1979.1056004.

2 Marshall Ball, Oded Goldreich, and Tal Malkin. Communication complexity with defective randomness. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 14:1–14:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.14.

3 Jop Briët, Harry Buhrman, Debbie W. Leung, Teresa Piovesan, and Florian Speelman. Round elimination in exact communication complexity. In Salman Beigi and Robert König, editors, *10th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2015, May 20-22, 2015, Brussels, Belgium*, volume 44 of *LIPIcs*, pages 206–225. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.TQC.2015.206.

4 Ran Canetti and Oded Goldreich. Bounds on tradeoffs between randomness and communication complexity. *Comput. Complex.*, 3:141–167, 1993. doi:10.1007/BF01200118.

5 Arkadev Chattopadhyay, Shachar Lovett, and Marc Vinyals. Equality alone does not simulate randomness. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 14:1–14:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CCC.2019.14.

6 Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 24–30. IEEE, 2020. doi:10.1109/FOCS46700.2020.00011.

7 Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM J. Comput.*, 24(4):736–750, 1995. doi:10.1137/S0097539792235864.

8 Rudolf Fleischer, Hermann Jung, and Kurt Mehlhorn. A communication-randomness tradeoff for two-processor systems. *Inf. Comput.*, 116(2):155–161, 1995. doi:10.1006/INCO.1995.1011.

9 Dale Jacobs, John Jeang, Vladimir Podolskii, Morgan Prior, and Ilya Volkovich. Communication complexity of equality and error correcting codes. Technical Report TR25-068, Electronic Colloquium on Computational Complexity (ECCC), 2025. URL: https://eccc.weizmann.ac.il/report/2025/068/.

10 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

11 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

12 Olivier Lalonde. Entanglement-assisted communication complexity and nonlocal games. *Master's Thesis, Université de Montréal*, 2023. URL: https://umontreal.scholaris.ca/items/d507c06f-33dc-49f5-ab91-31c2bd60c6bc.

13 Olivier Lalonde, Nikhil S Mande, and Ronald de Wolf. Tight bounds for the randomized and quantum communication complexities of equality with small error. *arXiv preprint arXiv:2107.11806*, 2021.

14 Bruno Loff and Sagnik Mukhopadhyay. Lifting theorems for equality. 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 50:1–50:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.STACS.2019.50.

15 Shay Moran, Makrand Sinha, and Amir Yehudayoff. Fooling pairs in randomized communication complexity. In Jukka Suomela, editor, *Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, volume 9988 of *Lecture Notes in Computer Science*, pages 49–59, 2016. doi:10.1007/978-3-319-48314-6_4.

16 Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991. doi:10.1016/0020-0190(91)90157-D.

17 Noam Nisan. The communication complexity of threshold gates. *Combinatorics, Paul Erdős Is Eighty*, pages 301–315, 1993.

18 Toniann Pitassi, Morgan Shirley, and Adi Shraibman. The strength of equality oracles in communication. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 89:1–89:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.89.

19 Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

20 Alexander A. Razborov. *Communication Complexity*, pages 97–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-19533-4_8.

21 Ron M. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

22 Tim Roughgarden. Communication complexity (for algorithm designers). *Found. Trends Theor. Comput. Sci.*, 11(3-4):217–404, 2016. doi:10.1561/0400000076.

23 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213. ACM, 1979. doi:10.1145/800135.804414.

A Appendix

A.1 Proof of Lemma 39

We start by illustrating the case of two iterations. In the first step, we apply the Reed-Solomon code with some parameters:

$$n \longmapsto (m_1, \varepsilon_1, \frac{n}{\varepsilon_1 2^{m_1}}),$$

and then instead of sending $\frac{n}{\varepsilon_1 2^{m_1}}$ bits, we apply the Reed-Solomon code (with new values of parameters) to this message again:

$$n_2 = \frac{n}{\varepsilon_1 2^{m_1}} \longmapsto (m_2, \varepsilon_2, \frac{n_2}{\varepsilon_2 2^{m_2}}).$$

Notice that in both iterations we use random bits, but only communicate according to the result of the final iteration. The resulting procedure for two iterations gives us the following parameters

$$n \longmapsto \left(m = m_1 + m_2, \varepsilon \leq \varepsilon_1 + \varepsilon_2, \frac{n}{\varepsilon_1 \varepsilon_2 2^{m_1+m_2}}\right),$$

and has the following restrictions

$$m_1 \leq \frac{n}{\varepsilon_1 2^{m_1}},$$
$$m_2 \leq \frac{n}{\varepsilon_1 \varepsilon_2 2^{m_1+m_2}}.$$

Observe that we are interested in ε and communication complexity only up to a constant factor. Further, notice that our errors are only additive from the first to the second iteration. Finally, we see that allowing a higher error rate in a given round will never increase the

amount of randomness or communication required. Therefore, we can assume that in each iteration we allow the same error ε; otherwise, we could simply choose to increase the allowed error in the iteration with a smaller error rate.

With this assumption, on an arbitrary iteration i we will have parameters

$$(m_i, \varepsilon, \frac{n}{\varepsilon^i 2^{m_1+\ldots+m_i}})$$

with the restriction

$$m_i \leq \frac{n}{\varepsilon^i 2^{m_1+\ldots+m_i}}. \tag{10}$$

The total values of parameters after c iterations (we assume that c is a constant) are

$$n \longmapsto \left(m = m_1 + \ldots + m_c, \varepsilon, \frac{n}{\varepsilon^c 2^{m_1+\ldots+m_c}}\right).$$

Recall that our protocol using single-iteration Reed-Solomon works only for $m < \frac{n}{\varepsilon 2^m}$. Thus our goal is to develop a protocol that is valid for larger values of m. Notice that only the final restriction directly contains a restriction on the total number of random bits used in the protocol:

$$m_c \leq \frac{n}{\varepsilon^c 2^{m_1+\cdots+m_c}} = \frac{n}{\varepsilon^c 2^m}. \tag{11}$$

Rearranging this expression, we get

$$2^m \leq \frac{n}{\varepsilon^c 2 m_c}.$$

We can see that minimizing the randomness used in the last iteration, m_c, while keeping m the same maximizes the range in which this protocol is valid. Naturally to minimize m_c, we need to maximize the randomness used in all previous iterations. Thus, to optimize the parameters we need to fix

$$m_i = \frac{n}{\varepsilon^i 2^{m_1+\ldots+m_i}}$$

on intermediate iterations.

We prove the following stronger version of Lemma 39.

▶ **Lemma 41.** *Recall that $f(x) = \log(\log \frac{1}{\varepsilon} + x)$. For any integer constant c we have that the communication protocol derived from c iterations of Reed-Solomon has complexity $\frac{n}{\varepsilon^c 2^m}$ and is valid for all*

$$m \leq \log n + c \log \frac{1}{\varepsilon} - f^{(c)}(\log n) \tag{12}$$

(up to an additive constant).

Moreover, if all restrictions (10) *are saturated, the inequality* (12) *is saturated as well.*

Proof. We prove this by induction on c. The base case corresponds to the standard Reed-Solomon protocol, which we already covered.

For the induction step, we do one extra step of the iteration and obtain the restriction

$$m_{c+1} \leq \frac{n}{\varepsilon^{c+1} 2^{m_1+\ldots+m_{c+1}}}.$$

Since for the previous iterations we need to maximize $m_1, \ldots, m_c$, we must fix them so that all restrictions (10) are saturated. By the inductive hypothesis we have that (12) is saturated. Substituting $m_1 + \ldots + m_c$ from this equality to above yields

$$m_{c+1} \leq \frac{2^{f^{(c)}(\log n)}}{\varepsilon 2^{m_{c+1}}}.$$

Rearranging this inequality we obtain, using Fact 6,

$$m_{c+1} \leq \log \frac{1}{\varepsilon} + f^{(c)}(\log n) - \log\left(\frac{1}{\varepsilon} + f^{(c)}(\log n)\right) = \log \frac{1}{\varepsilon} + f^{(c)}(\log n) - f^{(c+1)}(\log n).$$

Adding this to the expression for $m_1 + \ldots + m_c$ gives the desired inequality.

Moreover, it is easy to see that if (10) is saturated for $i = c + 1$ as well, the resulting inequality is saturated. ◀

Fairness and Efficiency in Two-Sided Matching Markets

Pallavi Jain
Indian Institute of Technology Jodhpur, India

Palash Jha
Indian Institute of Technology Jodhpur, India

Shubham Solanki
Indian Institute of Technology Jodhpur, India

Abstract

We propose a new fairness notion, motivated by the practical challenge of allocating teaching assistants (TAs) to courses in a department. Each course requires a certain number of TAs and each TA has preferences over the courses they want to assist. Similarly, each course instructor has preferences over the TAs who applied for their course. We demand fairness and efficiency for both sides separately, giving rise to the following criteria: (i) every course gets the required number of TAs and the average utility of the assigned TAs meets a threshold; (ii) the allocation of courses to TAs is envy-free, where a TA envies another TA if the former prefers the latter's course and has a higher or equal grade in that course. Note that the definition of envy-freeness here differs from the one in the literature, and we call it merit-based envy-freeness. We show that the problem of finding a merit-based envy-free and efficient matching is NP-hard even for very restricted settings, such as two courses and uniform valuations; constant degree, constant capacity of TAs for every course, valuations in the range $\{0, 1, 2, 3\}$, identical valuations from TAs, and even more. To find tractable results, we consider some restricted instances, such as, strict valuation of TAs for courses, the difference between the number of positively valued TAs for a course and the capacity, the number of positively valued TAs/courses, types of valuation functions, and obtained some polynomial-time solvable cases, showing the contrast with intractable results. We further studied the problem in the paradigm of parameterized algorithms and designed some exact and approximation algorithms.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Fair Matching, Envy-Freeness, Efficiency

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.38

Related Version *Full Version*: https://arxiv.org/abs/2504.10232 [24]

1 Introduction

Assigning teaching assistants (TAs) to courses is a practical challenge faced by many academic departments. The allocation should be fair for both instructors and TAs, taking into account their preferences and qualifications. For example, if a TA is willing to assist in a systems course, it is not desirable to assign them a theory course. On the contrary, it may also happen that a TA might like to assist in a course (maybe because their friends are assisting in that course or course is less loaded), however he does not have enough knowledge in that course. Considering some of these critical requirements of instructors and TAs, our department follows the following protocol: we ask TAs to submit their preferences over the courses they would like to assist with and their grades in those courses, and then instructors are asked to submit their preferences over the TAs who are interested in their courses. We try to ensure that every course gets the required number of TAs (at least some of the choice of instructor) and prioritise the choice of the TAs with higher grades. However, finding a

© Pallavi Jain, Palash Jha, and Shubham Solanki;
licensed under Creative Commons License CC-BY 4.0

satisfactory allocation that meets the requirements of both sides is not trivial. In this paper, we address this problem from a theoretical perspective, using tools from algorithms and computational social choice theory.

1.1 Mathematical Formulation

We have a set of courses $X = \{x_1, \ldots, x_n\}$ and a set of TAs $T = \{t_1, \ldots, t_m\}$. Every TA has a utility for every course, i.e., for every $i \in [m]$, we have a utility function $u_i \colon X \to \mathbb{Z}_{\geq 0}$. Furthermore, every TA t_i has a grade function $g_i \colon X \to \mathbb{Q}_{\geq 0}$. For simplicity, we assumed that every TA has done all the courses (grade can be considered 0 if a TA has not credited a course). Every course $x_i \in X$ has a capacity c_i (that captures the required number of TAs), and a utility function $v_i \colon T \to \mathbb{Z}_{\geq 0}$. If a TA t has utility 0 for the course x_i, then $v_i(t) = 0$, otherwise $v_i(t) > 0$. This is a valid assumption, as we would like to respect the choices of a TA. A matching $\mu \colon T \to X \cup \{\emptyset\}$ is *feasible* if for every course $x_i \in X$, $|\mu^{-1}(x_i)| = c_i$, i.e., c_i TAs are assigned to the course x_i, and no TA is assigned to a zero-valued course. We call that a matching $\mu \colon T \to X \cup \{\emptyset\}$ is *fair* if it is feasible and for a given $k \in \mathbb{Q}_{\geq 0}$, it satisfies the following two conditions.

- **Satisfaction of Courses.**[1] For every course $x_i \in X$, $\mathsf{AvgUtil}(x_i) = \frac{\sum_{t \in \mu^{-1}(x_i)} v_i(t)}{c_i}$ is at least k.
- **Merit-based envy-freeness between TAs.** A TA t_i envies another TA t_j on merit basis if $g_i(\mu(t_j)) \geq g_j(\mu(t_j))$ and $u_i(\mu(t_j)) > u_i(\mu(t_i))$, i.e., grade of t_i in the course allocated to t_j is at least the grade of t_j in that course and t_i values the course allocated to t_j more than their own allocated course. In the matching μ, there should not be any pair of merit-based envious TAs.

We call this fair matching as *merit-based envy-free egalitarian* (MEFE) matching, and the problem of finding such a matching as MEFE-Matching.

Highest grade vs highest utility in a course. It seems desirable to assume that the highest grader in a course will have the highest utility. However, this need not be the case. There might be other responsible factors for course utilities. For example, if a TA applicant has previously done a relevant course or project under the course instructor who trusts the TA's expertise in the area and soft skills, the instructor would probably assign a higher utility for that TA. A TA applicant with prior experience at TAship or a recommendation from other instructors might also be assigned higher utility due to their experience. While TAs might have done the course previously, they may have done it in different universities. In such a case, it is hard to gauge a TA accurately just based on their grades. Thus, we demand utility functions from the courses separately.

1.2 Significance of our Fairness Notions

The deferred-acceptance algorithm (commonly-used algorithm in two-sided markets) is biased towards one side of the agents [14], which is not desirable for our application. Thus, we move our attention to some of the well-studied fairness criterion used in the field of *fair allocation*.

On the course side, we considered criteria similar to *egalitarian welfare*, but we take the average for the following reason: suppose that there is a course x with capacity 10 and a course y with capacity 1. Now, if we require a threshold for satisfaction, say 10, then the

[1] This is efficiency criteria. But, we include it in the definition of fair matching for brevity.

course y gets a very good TA; however, course x may get TAs each with utility 1. To avoid such a predicament, we considered average. Since we require good TAs in all the courses, we think *egalitarian type* criteria is a better choice over other welfare functions such as utilitarian or Nash. We believe that envy-freeness (the most popular fairness criteria in fair allocation) is not suitable here for the following reason: what if everybody gets a bad set of TAs? Nobody may envy anybody, but this is a disastrous situation for everyone. We also believe that instructors care more about their TAs instead of comparing them with the TAs of other courses.

On the TAs side, we considered *envy-freeness type* fairness. As discussed earlier, it is important to consider grades as well. Indeed if a matching is envy-free, then it is also merit-based envy-free. However, it is possible that there exists a merit-based envy-free matching even if envy-free matching does not exist. Consider a classical example of non-existence of envy-free solution, when there is only one course x with capacity 1, and two TAs t, t', both value x positively. This does not have envy-free solution (irrespective of actual valuations). But, if x values t (say $\geq k$) more than t' and the grade of t is more than t', then we match t with x, and it is a merit-based envy-free solution. As discussed earlier, conceptually also, it is crucial to consider grades.

Since grades need to be considered, egalitarian criteria is not interesting for TAs.

Interestingly, MEFE matching is also a weakly stable matching when course satisfaction is not under consideration or utility of course x for TA t is same as the grade of t in the course x, however the converse need not be true. For contradiction, we assume that the MEFE-Matching solution is not weakly stable. Let the blocking pair be (x_i, t_j). Then t_j prefers x_i over $\mu(t_j)$ and has a higher grade in x_i than $\mu^{-1}(x_i)$. Hence t_j envies $\mu^{-1}(x_i)$, a contradiction. The converse may not be true. For example, consider an instance with 1 course and 2 TAs both having the same grade in the course, with the course having capacity 1. Any assignment will be weakly stable as the course is indifferent between the two TAs but the unassigned TA will always envy the assigned TA. Additionally, if the course values both TAs less than k, no assignment would lead to satisfaction of the course.

It can be seen that merit-based envy-freeness exactly aligns with justified envy-freeness (as, for instance, characterized by Yokoi in [39]) when course utility for a TA equals the grade of the TA in that course, with k equalling 0, all TAs assign distinct utilities to courses, and no two TAs have the same grade in a course. Therefore, MEFE-Matching serves as a generalization of the justified envy-freeness model.

Indeed, there are several notions of fairness in the fair allocation literature, which will be interesting to consider, and some are already considered (see Related Work).

1.3 Broader applications of MEFE-Matching

We discuss two other practical applications of MEFE-Matching. The MEFE-Matching model might be applied in a job market with a mediator company. Job applicants (analogous to TAs in our model) hold utilities over the firms (analogous to courses in our model) that the mediator company has ties with. Each of these firms might hold tests (grades in a course) specific to them, conducted by the mediator company. However, the firms assign utilities to applicants based on resume shortlisting, HR rounds, and test scores. Here, applicants might envy each other based on their preferences (utilities assigned by TAs) and test scores for companies. And firms would want to ensure that they are satisfied with the applicants that are allocated to them.

We could also use MEFE-Matching in Medical Residency Matching. Residency applicants (TAs) hold utilities over residencies (courses). Additionally, applicants might have grades in related areas to the residencies they are applying to. Residencies hold utility over applicants based on performance in relevant areas, clinical experience, and interviews.

Table 1 Summary of results where color denotes the ties in TA's preference list. Green means ties are allowed and red means ties are not allowed. Text written in blue denotes the condition that in a course no two TAs should have the same grade. The number of courses and TAs are denoted by n and m, respectively. "-" means the values can be arbitrary. **In the approximation result (last row), $\mathsf{max}_{\mathsf{val}}$ is a function of n.**

# courses	Capacity	Degree	Types of valuations	Result	Theorem
2	-	of TA $= 2$	by TA $= 1$	NP-hard	[1]
-	1	of TA / course ≤ 3	by course $= 1$	NP-hard	[2]
-	1	-	-	polynomial	[7]
-	2	of TA ≤ 3, course ≤ 6	by course $= 3$,by TA $= 1$	NP-hard	[4]
-	c	of course $\leq c+1$	-	polynomial	[5]
-	-	of TA $= 1$	-	polynomial	[8]
-	-	-	by course $= 2$	polynomial	[11]
constant	constant	-	-	polynomial	[10]
-	constant	-	by course $=$ constant	FPT(n)	[13]
-	-	-	-	FPT(m)	[12]
-	constant	-	-	$(1-\epsilon)$- approximation in FPT(n, ϵ)	[14]

1.4 Our Contribution

Our model considers a fairness criterion for TAs and an efficiency criterion for courses. This is motivated by the fact that the requirements of both sides in matching under two-sided preferences need not be the same. This is our first conceptual contribution.

Next, we move to our technical contribution. Table 1 summarises our complexity results. Some of the technical highlights of our work are discussed below. Throughout the paper, by *degree* of a course, we mean the number of TAs positively valued by the course. Similarly, we define the degree of a TA, as the number of courses positively valued by a TA.

Hardness Results

Unsurprisingly, similar to many problems in Fair Allocation, the problem is NP-complete for two courses (**Theorem 1**), due to a reduction from the Equal-Cardinality Partition problem. In this reduction, the degree of courses is "large", however the degree of TAs is two. Furthermore both the courses have same valuation functions and all the TAs equally value all the courses, in fact, their grade is also same in all the courses. We next ask the question: *does the large degree of courses lead to intractability?* Surprisingly, the problem is NP-complete even when the degree of courses is three (**Theorem 2**). In the same reduction, the degree of TAs is also constant, in particular, three. In fact, the capacity of all the courses is one and each course values all positively valued TAs equally (refer to the theorem statement for more restrictions). In light of Theorem 2, we next ask: *do the different valuations of TAs lead to intractability*? **Theorem 4** answers this question negatively. The next set of results, highlights more about the inherent nature of instance that leads to intractability.

Polynomial Time Algorithms

We observe that the large degree of courses is not responsible for the intractability; rather, the difference between degree and capacity is one of the responsible factors. In Theorem 2, the difference between degree and capacity is two. In **Theorem 5**, we show that the problem is polynomial-time solvable when this difference is at most one (the actual value of degree and capacity does not matter). This result also resolves the complexity when the degree of courses is two (**Corollary 6**), contrasting Theorem 2. We further observed that when the capacity of each course is one, then the same valuations of a TA for multiple courses

lead to intractability, which is easy to avoid, but if all TAs value positively valued courses differently, then it leads to tractability (**Theorem 7**). Due to Theorem 1, we have hardness when the number of courses is two and the degree of TAs is two. When we relax one of these constraints, i.e., the degree of TAs is one or there is only one course, it leads to tractability (**Theorem 8 and Corollary 9**). In all our hardness results, either the capacity of courses is constant, or the number of courses is constant. When both are constant, then it leads to a tractability result (**Theorem 10**). Contrasting Theorem 4, **Theorem 11** shows that the problem is polynomial-time solvable when every course has at most two distinct positive valuations for all TAs, but, here we require that for a course no two TAs have the same grade (which is not a very strict restriction if we consider actual marks).

Parameterized (Approximation) Algorithms

MEFE-Matching has some natural parameters to consider: the number of TAs (m), then number of courses (n), the maximum degree of courses/TAs ($\mathsf{d}_{\mathsf{course}}, \mathsf{d}_{\mathsf{TA}}$), maximum capacity of a course (cap), the number of distinct valuation/grade functions ($\mathsf{type}_{\mathsf{val}}, \mathsf{type}_{\mathsf{grade}}$), maximum value that a valuation/grade functions can take ($\mathsf{max}_{\mathsf{val}}, \mathsf{max}_{\mathsf{grade}}$). **Theorem 12** shows positive results with respect to m. Due to Theorem 2, we have paraNP-hardness with respect to $\mathsf{d}_{\mathsf{course}}+\mathsf{d}_{\mathsf{TA}}+\mathsf{type}_{\mathsf{val}}+\mathsf{type}_{\mathsf{grade}}+\mathsf{max}_{\mathsf{val}}+\mathsf{max}_{\mathsf{grade}}+\mathsf{cap}$. Thus, we cannot hope for a *fixed-parameter tractable*(FPT) algorithm with respect to the combination of these parameters. We design an FPT algorithm with respect to n when $\mathsf{type}_{\mathsf{val}}$ and cap are constant for courses, no TA values two courses equally and no course has same grade for two TAs (**Theorem 13**). When we relax the constraint of constant $\mathsf{type}_{\mathsf{val}}$ (i.e., it is no longer constant) and $\mathsf{max}_{\mathsf{val}}$ is a function of n, then we obtain a $(1-\epsilon)$-approximation algorithm that approximates satisfaction of every course and runs in $\mathsf{FPT}(n, \epsilon)$ (**Theorem 14**). Note that due to Theorem 1, we cannot hope for an FPT algorithm with respect to $n + \mathsf{degree}_{\mathsf{TA}} + \mathsf{type}_{\mathsf{val}} + \mathsf{type}_{\mathsf{grade}}$.

Existence of Solution

In Section 7, we identify two yes-instances and some no-instances of the problem.

1.5 Related Work

The literature on matching under two-sided preferences is vast. It originated from the seminal work of [14] on the stable matching problem. The unfairness of the deferred-acceptance algorithm, together with the immense practical applicability of stable matchings, has generated considerable interest in developing algorithms for finding *fair* stable matchings. Several fairness concepts have been studied in conjunction with stability, including *strong and super stability* [21], *minimum regret* [28], *egalitarian* [33, 22],[2] *median* [38, 37], *sex-equal* [19], *balanced* [11, 18], *leximin* [35], and Nash [25].

For the related work on matching on one-sided preferences (a field of Fair Allocation), we refer the reader to recent surveys [1, 31, 34]. For two-sided preferences, we refer the reader to the following literature [6, 2, 32].

Freeman et al. [13] consider envy-freeness up to one good and maximin share guarantee under two sided-preferences. But, in their model, the fairness criteria are the same on both sides. However, this is the first work (to the best of our knowledge) that borrows fairness

[2] In the stable matching literature, the term *egalitarian matching* has been used for matchings that maximize the total satisfaction of the agents by minimizing the *sum* of ranks of the matched partners [33, 22]. This objective is different from *egalitarian welfare*, which maximizes the utility of the least-happy agent, which we considered.

notions from the fair division literature to the two-sided preferences, conceptually closer to our work. Bu et al. [5] extended this work and considered the fairness criterion envy-freeness up to c-good and proportionality up to c-good. They also considered the same fairness criteria on both the sides. Patro et al. [36] considered envy-freeness up to one individual on one side and maximin share on the other side. Gollapudi et al. [16] also considered envy-freeness up to one individual on both the sides along with the maximum weight of the matching in case of repeated matchings. Igarashi et al. [20] considered stability condition along with envy-free up to one individual for one side of the agents. Recently, Sung-Ho Cho [7] studied the problem from a mechanism design perspective and considered envy-freeness up to k-peers. There are several other papers that study fairness in two-sided preferences (not directly related to our work) [27, 10, 26, 3, 17, 29]. Indeed the list is not exhaustive.

Bredereck et al. [4] also studied local envy-freeness in house allocation problems where each agent receives only one item. Our model generalizes local envy-freeness as follows. We create a directed graph for each course, where TA t has an arc to TA t' if t has a grade at least as high as t' in that course. We seek an envy-free allocation from the TAs' perspective that meets this condition: if course x is assigned to t, let Z be the set of courses t values more than x. We then check the local envy-free condition for t in all graphs corresponding to the courses in Z.

2 Preliminaries

Throughout the paper, $(X, T, \{v_i\}_{i\in X}, \{u_i\}_{i\in T}, \{g_i\}_{i\in T}, \{c_i\}_{i\in X}, k)$ denotes an instance of MEFE-Matching, where $X = \{x_1, \ldots, x_n\}$ and $T = \{t_1, \ldots, t_m\}$. In general, we assume that $m \geq n$, since each course has capacity at least 1. If $m < n$, we will have a trivial no-instance. When referring to an element $t \in T$, without a subscript, we may use grade function $g_t : X \to \mathbb{Q}_{\geq 0}$, and utility function $u_t : X \to \mathbb{Z}_{\geq 0}$, both with subscript t. Similarly, when referring to a course $x \in X$, without a subscript, we may use utility function $v_x : T \to \mathbb{Z}_{\geq 0}$. By *degree* of a course, we mean the number of TAs positively valued by the course. Similarly, we define the degree of a TA, as the number of courses positively valued by the TA. We denote the degree of course/TA z by $d(z)$. For a course $x \in X$, $N(x)$ is the set of all TAs positively valued by x. Similarly, for a TA $t \in T$, $N(t)$ is the set of all courses positively valued by t. We define a similar notion for $Z \subseteq X$ (or $Z \subseteq T$) as follows: $N(Z) = \cup_{z\in Z} N(z)$.

By *types of valuations* of a course/TA z, we mean the number of distinct positive values assigned by z to TAs/courses. The functions $f\colon P \to R$ and $g\colon P \to R$ are *identical*, if for all $z \in P$, $f(z) = g(z)$. A function whose range is $\{0, 1\}$ is called *binary*. For $z \in X \uplus T$, we also use the notation u_z for the utility or grade function of z.

Let μ be a matching. If a course (TA) is not matched to a TA (course) in T (X), then we call it *unassigned* or *unsaturated* with respect to μ; otherwise it is called *assigned* or *saturated*.

For an instance $\mathcal{I}$ of MEFE-Matching, let $G^{\mathcal{I}} = (X, T)$ be a bipartite graph such that $E(G^{\mathcal{I}}) = \{(x, t)\colon u_t(x) \neq 0\}$. Note that $G^{\mathcal{I}}$ can be a disconnected graph. We skip the subscript $\mathcal{I}$, if the instance is clear from the context.

In parameterized algorithms, given an instance I of the problem Π, and an integer k (called a parameter), the goal is to design an algorithm that runs in $f(k) \cdot |I|^{\mathcal{O}(1)}$ time, where f is an arbitrary computable function depending on the parameter k. Such algorithms are known as FPT algorithms. If there exists an FPT algorithm with respect to the parameter k for the problem Π, we say that "Π can be solved in FPT(k)".

All omitted proofs are in the full version of the paper [24]. For more details on the subject in general, we refer to the textbooks [8, 12, 9].

3 Hardness Results

In this section, we prove the intractability of the problem even for very restricted cases. Given a matching μ, checking whether the feasibility constraint is satisfied can be done in polynomial time. Verifying the satisfaction constraint for a course c_i takes $\mathcal{O}(c_i)$ time. Finally, to check the merit-based envy-freeness condition for a TA t_i, we compare the grades of all TAs matched to courses preferred by t_i over its assignment, which can be done in polynomial time. Thus, the problem is in NP.

Our first intractability result is for two courses with identical valuations, where degree of TAs is two, contrasting Theorem 8 and Corollary 9.

The result is due to the polynomial-time reduction from the Equal-Cardinality Partition problem, in which given a multiset $\mathcal{S}$ of positive integers, the goal is to partition $\mathcal{S}$ into two subsets of the same size that have the same sum. This problem is known to be NP-complete [15].

▶ **Theorem 1 (♣).**[3] *MEFE-Matching is NP-complete even when there are two courses with identical valuation functions; grades and valuations from TA side are* 1.

Note that if we constraint the types of valuation by the courses to be a constant, r (order $\mathcal{O}(1)$), in addition to the constraints in Theorem 1, the problem becomes polytime. Indeed, since each course can assign TAs according to only one of r valuation types, with grades and valuations from TA side being 1, we can simply enumerate all possible combinations of assignments of TAs to the 2 courses in time given by $\mathcal{O}(m^r)$. We can then check whether any of these assignments leads to a MEFE matching in polynomial time.

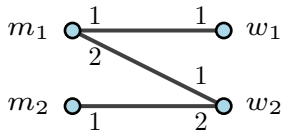
Next, we show that the problem is NP-complete even when the degree of courses and TAs is three, and capacity of each course is one. The result is due to a polynomial-time reduction from (3-3)-com smti, which is known to be NP-complete [23, Theorem 5].

▶ **Theorem 2.** *MEFE-Matching is NP-complete even when $v_j(t) = g_t(x_j)$ for all $t \in T, x_j \in X$, i.e., utility function is derived from grades and the capacity of each course is one; each TA has degree at most three, valuations and grades are in $\{0, 1, 2, 3\}$, and no TA has same grade for any two courses. It is also NP-hard when each course has binary valuation function, with all other constraints remaining the same.*

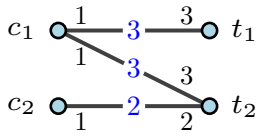
Proof. We give a polynomial-time reduction from the (3-3)-com smti problem. In (3-3)-com smti, we have sets M, W of men and women. For $x \in M \uplus W$, let pref_x denote the preference list of x. We assume that $x \in M$ without loss of generality (i.e., the following constraints hold for the preference lists of each woman in (3-3)-com smti as well). The list pref_x is an ordered sequence of disjoint subsets of women which represents a partial order over the women, where women in earlier subsets are preferred over women in later subsets, and women in the same subsets are tied. We define $|\mathsf{pref}_x|$ as the total number of women appearing in pref_x, and (3-3)-com smti requires that $|\mathsf{pref}_x| \leq 3$. Women which do not appear in pref_x are considered unacceptable by x. In (3-3)-com smti, the goal is to find a weakly stable matching of men and women, given the above constraints.

Let $\mathcal{J}$ be an instance of (3-3)-com smti such that the ties are only on the women's side. We construct an instance $\mathcal{I}$ of MEFE-Matching as follows. Let $X = M$ and $T = W$, i.e., for each man $m \in M$, we have a course m, and for each woman w, we have a TA w. For every course $m \in X$, $v_m(w) = 4 - i$, if man m ranks woman w at position i; otherwise 0. We next define a set of valuation functions of TAs. It is according to the preference list of

[3] Proof of all the theorems/lemmas/claims marked with ♣ can be found in the full version of the paper [24].



(a) An instance of (3-3)-COM SMTI.



(b) An instance of MEFE-MATCHING where grades are in the middle of the edges in blue color.

Figure 1 An instance of MEFE-MATCHING is presented in 1b corresponding to an instance of (3-3)-COM SMTI presented in 1a.

women. If woman w ranks man m at position i, then the utility of TA w for the course m is $u_w(m) = 4 - i$, if m is not in the preference list of woman w, then $u_w(m) = 0$. Next, we define a grade function for every course based the preference list of men, which is strict. If man m ranks woman w at position i, then the grade of TA w for the course m is $g_w(m) = 4 - i$, if m is not in the preference list of woman w, then $g_w(m) = 0$. The capacity of each course is one and $k = 1$. Next, we prove the correctness. In particular, we prove the following.

▶ **Lemma 3.** *$\mathcal{J}$ is a yes-instance of (3-3)-COM SMTI if and only if $\mathcal{I}$ is a yes-instance of MEFE-MATCHING.*

Proof. In the forward direction, let η be a solution to $\mathcal{J}$. We claim that $\mu = \eta$ is also a solution to $\mathcal{I}$. Clearly, μ is a feasible matching as every course is matched to one TA. Furthermore, for every course $m \in X$, $\mathsf{AvgUtil}(m) \geq 1$ as $v_m(w) \geq 1$ for all w who are in preference list of m and if a woman w is matched to a man m then w must be in the preference list of m. Next, we argue that μ is merit-based envy-free. Note that μ is a perfect matching. Thus, every TA is matched to a course. Furthermore, every TA is matched to a course with non-zero utility due to the construction and the fact that η is a solution to (3-3)-COM SMTI. Suppose that TA t envies TA t', then we have $g_t(\mu(t')) \geq g_{t'}(\mu(t'))$. Due to construction and the fact that ties belong to only women side, equality is not possible. Hence, $g_t(\mu(t')) > g_{t'}(\mu(t'))$. Also, $u_t(\mu(t')) > u_t(\mu(t))$ which means $t \succ_{\eta(t')} t'$ and $\eta(t') \succ_t \eta(t)$, which contradicts the fact that η is a stable matching.

In the reverse direction, let μ be a solution to $\mathcal{I}$. We claim that $\eta = \mu$ is a solution to $\mathcal{J}$. Recall that the capacity of every course is one in $\mathcal{J}$. Thus, η saturates all men. Since, the number of courses and TAs is same and every course is matched to only one TA, every TA is matched to a course in μ. Thus, η is a perfect matching. Moreover, for every course $w \in X$, $\mathsf{AvgUtil}(w) \geq 1$ as we have $v_m(w) \geq 1$ for matched pair (m, w). Each man is matched with a woman who is in his preference list and hence, each woman also is matched with a man in her preference list due to construction. Next, we claim that μ is weakly stable. Towards the contradiction, suppose that there is a pair (m, w) such that $w \succ_m \eta(m)$ and

$m \succ_w \eta(w)$. Thus, due to the construction of the instance $\mathcal{I}$, $g_w(m) > g_{\eta(m)}(m)$, and $u_w(m) > u_w(\mu(w))$. Thus, TA w envies TA w', which contradicts that μ is a solution to $\mathcal{I}$. ◀

The transformation can be carried out in polynomial time since, for each pair $(m, w) \in M \times W$, the values $v_m(w)$, $u_w(m)$, and $g_w(m)$ can be computed by scanning the preference lists of m and w, each of which has length at most $\mathcal{O}(n)$. Thus, the total time required to construct the instance is $\mathcal{O}(n^2)$. This completes the NP-hardness proof when course utilities are derived from grades. The binary case is similar. Here, the course valuation would be defined as follows: for every course $m \in X$, $v_m(w) = 1$ if woman w is in the preference list of man m, otherwise 0. The proof remains same for binary valuation function. ◀

Our next intractability result is when TAs have binary valuations and grades in the range $\{0, 1, 2\}$, other than the constraints on degrees and capacities. The result is due to a polynomial-time reduction from the 3-DIMENSIONAL PERFECT MATCHING problem.

▶ **Theorem 4 (♣).** *MEFE-MATCHING is NP-complete even when each course has capacity two, degree six, and valuations in* $\{0, 1, 2, 3\}$*; each TA has binary valuation function, and grades in* $\{0, 1, 2\}$*.*

4 Polynomial Time Tractable Cases

In this section, we identify some instances that can be solved in polynomial time. We first consider the case when the difference of the degree and capacity of each course is at most one.

▶ **Theorem 5.** *MEFE-MATCHING can be solved in* $\mathcal{O}(m^2n^2)$ *when the difference between the degree and capacity of each course is at most one.*

Proof. We visualize an instance $\mathcal{I}$ of MEFE-MATCHING as a bipartite graph $G = (X, T)$ and note that G can be a disconnected graph. However, it is sufficient to solve each component separately, as we do not assign 0-valued TAs to a course. Let G_{conn} be a maximal connected subgraph of G. For the ease of notation, we reuse X, T as the bipartiton of G_{conn} as well, where X, T is the set of courses and TAs, respectively, in G_{conn}. Let $X = \{x_1, \ldots, x_n\}$ and $T = \{t_1, \ldots, t_m\}$. We find an MEFE matching, if it exists, for G_{conn}. We design the algorithm based on the structure of the graph. We first consider the case the graph has a course with degree equalling capacity. If not, then we further consider other cases: whether it is acyclic, or contains one cycle, or contains at least two cycles. In the case of two cycles, we argue that it is no-instance. In the other cases, for a yes-instance, the intuitive idea is that there exists a TA/course whose assignment or decision that it is unassigned dictates the whole matching. Note that such a TA/course should be chosen carefully as every choice does not fix the matching.

Case 1: There exists a course x_i such that $d(x_i) - c_i = 0$. Let $X' = \{x_1, \ldots, x_\ell\}$ be the set of courses in X such that $d(x_i) = c_i$, where $i \in [\ell]$. We construct a matching μ as follows: for each $i \in [\ell]$, $\mu^{-1}(x_i) = N(x_i)$. If $\mu^{-1}(x_i)$ and $\mu^{-1}(x_j)$ are not disjoint, for any $i, j \in [\ell]$, then we return NO. Otherwise, we call `Extended Matching`$(G_{conn}, \mu, T_\mu = N(X'), X_\mu = X')$(Algorithm 1) and find a matching μ. The intuition is that we consider a course $x \in X$ whom we did not assign TAs yet, such that one of the positively valued TAs it has an edge with has already been assigned to a course. Thus, all the remaining TAs positively valued by this course must be assigned to it (we will argue in the correctness proof that for a yes-instance, only one of its TA is assigned to another course). If μ is a solution to $\mathcal{I}$, we return it; otherwise, we return NO.

Case 2: For all the courses $x_i \in X$, $d(x_i) - c_i = 1$.

Case 2.1: G_{conn} is acyclic, i.e., a tree. In this case, $|E(G_{conn})| = |X| + |T| - 1$. Furthermore, since G_{conn} is a bipartite graph, $E(G_{conn}) = \sum_{i\in[n]} d(x_i)$. Thus,

$$\sum_{i\in[n]} d(x_i) = |X| + |T| - 1$$

This implies that $\sum_{i\in[n]}(d(x_i) - 1) = |T| - 1$ Since $c_i = d(x_i) - 1$, we have that

$$\sum_{i\in[n]} c_i = |T| - 1$$

Algorithm 1 `Extended Matching`.

Input: a connected graph $G_{conn} = (X, T)$, a matching μ, a set of TAs T_μ, a set of courses X_μ
Output: a matching μ

while $\exists x \in N(T_\mu)$ such that $x \notin X_\mu$ **do**
  for all TA $t' \in T \setminus T_\mu$ such that $xt' \in E(G_{conn})$ **do**
    $\mu(t') = x$
    $T_\mu \to T_\mu \cup \{t'\}$
  $X_\mu \to X_\mu \cup \{x\}$
return μ

Hence, there is only one TA that is unassigned in any feasible solution to $\mathcal{I}$. We guess this TA, say t. Now, we need to assign all TAs in $T' = T \setminus \{t\}$ to the courses in X. The intuitive idea is that since t is unassigned, for all the courses that have positive valuations to t, we only have capacity many choices. Thus, the allocation of all these courses is fixed. Since the graph is connected, this eventually fixes the allocation of all the courses. Algorithm 2 describes the algorithm formally.

Algorithm 2 Unit difference between degree and capacity: Acyclic Case.

Input: a connected graph $G_{conn} = (X, T)$
Output: either a matching μ, or NO.

for $t \in T$ **do**
  $T_{\mu_t} = t$, $X_{\mu_t} = \emptyset$, $\mu_t(z) = \emptyset$, for all $z \in T$
  $\mu_t =$ `Extended Matching`$(G_{conn}, \mu_t, T_{\mu_t}, X_{\mu_t})$
  if μ_t is a solution to $\mathcal{I}$ **then return** μ_t
return NO

Case 2.2: G_{conn} contains only one cycle. Let $C = (x_1, t_1, \ldots, x_\ell, t_\ell)$ be the cycle in G_{conn}. Clearly, there exists an edge in C whose deletion makes the graph G_{conn} acyclic. Thus, in this case, $|E(G_{conn})| = |X| + |T|$. Furthermore, $|E(G_{conn})| = \sum_{i \in [n]} d(x_i)$. Thus, by equating the above two expressions just as we did in Case 2.1, we get $|T| = \sum_{i \in [n]} c_i$. Therefore, all TAs need to be assigned in a feasible matching for this case. Let X_C, T_C denote the set of courses and TAs, respectively, in C. We guess whether t_1 or t_ℓ is assigned to x_1 (one of them has to be assigned as $d(x_1) - c_1 = 1$). After this guess, an allocation for all the courses gets "fixed", i.e., there is a unique choice of allocation. Algorithm 3 describes the algorithm formally.

Case 2.3: G_{conn} contains more than one cycle. The algorithm returns NO in this case.

The detailed proof of correctness and running time analysis can be found in the full version [24]. ◀

Due to Theorem 5, we have the following result, which is in contrast to Theorem 2 when considering the degree of courses.

▶ **Corollary 6.** *MEFE-Matching can be solved in polynomial time when the degree of courses is at most two.*

Our next tractability result is in contrast to Theorem 2, as it reduces the problem to Weighted Strongly Stable Matching with Ties and Incomplete Lists (WSSMTI) in polynomial time, which can be solved in polynomial time [30].

Algorithm 3 Unit difference between degree and capacity: Unique cycle case.

Input: a connected graph $G_{conn} = (X, T)$
Output: either a matching μ, or NO.

let $C = (x_1, t_1, \ldots, x_\ell, t_\ell)$
let $\mu^{-1}(x_1) = N(x_1) \setminus \{t_\ell\}$
$X_\mu = \{x_1\}$, $T_\mu = N(x_1) \setminus \{t_\ell\}$,
μ=`Extended Matching`$(G_{conn}, \mu, T_\mu, X_\mu)$
if μ is a solution to $\mathcal{I}$ **then return** μ
$\tilde{\mu}^{-1}(x_1) = N(x_1) \setminus \{t_1\}$
$\tilde{\mu}^{-1}$=`Extended Matching`$(G_{conn}, \tilde{\mu}^{-1}, T_\mu, X_\mu)$
if $\tilde{\mu}^{-1}$ is a solution to $\mathcal{I}$ **then return** μ'
return NO

▶ **Theorem 7 (♣).** *MEFE-Matching can be solved in $\mathcal{O}(nm^2)$ when the capacity of each course is one, and each TA has distinct valuations for positively valued courses.*

Our next result is in contrast to Theorem 1. The result is due to the observation that a TA with high grade needs to be prioritised over low grade TAs in a solution. Thus, we sort the TAs based on their grades and match the top capacity many TAs to the course. We also observe that an instance can be partitioned into distinct sub-instances, where each sub-instance has only one course.

▶ **Theorem 8 (♣).** *MEFE-Matching can be solved in $\mathcal{O}(m \log m)$ when the degree for each TA is one.*

▶ **Corollary 9.** *MEFE-Matching can be solved in polynomial time when the number of courses is one.*

Proof. The degree of every TA is 1 when the number of courses is one. Hence Theorem 8 is applicable. ◀

Another restricted instance we analyze is one in which the capacities of all courses are constant, and the number of courses is also constant. In this case, the total number of possible solutions are polynomial which can be enumerated in polynomial time. Hence, we have the following result.

▶ **Theorem 10 (♣).** *MEFE-Matching can be solved in $\mathcal{O}(m^{\mathcal{O}(1)})$ when the number of courses and capacity of each course is constant.*

Next, we consider the case when every course has only two positive distinct values for TAs, and obtain the tractability, with some restrictions on valuations and grades of TAs.

▶ **Theorem 11.** *MEFE-Matching can be solved in $\mathcal{O}(\max(nmlog(m), m^2))$ when every course has only two distinct positive valuations, and for a course no two TAs have the same grade. Furthermore, each TA has distinct valuations for positively valued courses.*

Proof. Let $\mathcal{I} = (X, T, \{v_i\}_{i \in X}, \{u_i\}_{i \in T}, \{g_i\}_{i \in T}, \{c_i\}_{i \in X}, k\}$ be an instance of MEFE-Matching such that $v_i \colon T \to \{0, q_i, q_i'\}$, where q_i, q_i' are positive integers, and for any pair of TAs $t_i, t_j \in T$, $g_i(x) \neq g_j(x)$, for any course x. Without loss of generality, let $q_i \geq q_i'$. Next, we find the number of q_i and q_i' valued items assigned to x_i in a solution. Let

a_i, a'_i be the number of q_i and q'_i valued items assigned to x_i, respectively. If $q_i = q'_i$, then $a_i = c_i, a'_i = 0$. Otherwise, we find the following equations for each course independently. Due to our feasibility and satisfaction constraints for courses, we know that for each $i \in [n]$,

$$a_i q_i + a'_i q'_i \geq k c_i \tag{1}$$

$$a_i + a'_i = c_i \tag{2}$$

If the set of the above two equations does not have a solution, then clearly, $\mathcal{I}$ is a no-instance of MEFE-Matching. Suppose that there is a solution of the set of Equation 1 and 2 for every course. If we solve these two equations, we obtain that $a'_i \leq \frac{c_i(q_i-k)}{q_i-q'_i}$ and $a_i \geq \frac{c_i(k-q'_i)}{q_i-q'_i}$. Note that if a'_i is negative, then $a_i > c_i$, which violates the feasibility of a matching. Thus, if $q_i < k$, it is a no-instance. So next we assume that $q_i \geq k$. We choose $a_i = \max\{0, \lceil \frac{c_i(k-q'_i)}{q_i-q'_i} \rceil\}$ and $a'_i = c_i - a_i$. We next give a polynomial time reduction from MEFE-Matching to the Stable Matching problem for bipartite graphs, in which given a set of men, say M, and women, say W, a preference list of women over men, a preference list of men over woman, (preference lists might be incomplete); the goal is to decide whether there exists a matching such there is no pair of man and woman (m, w) such that (i) m is unmatched or prefers w over his matched partner, and (ii) w is unmatched or prefers m over her matched partner. Such a pair of (m, w) is called a *blocking pair*, and a matching without blocking pairs is called a *stable matching*.

We create an instance of the Stable Matching problem $\mathcal{J}$ as follows. The set of men is $M = T$, i.e., corresponding to every TA in T, we have a man in M. Corresponding to every course $x_i \in X$, we have c_i women, say $w_i^1, \ldots, w_i^{c_i}$. If $a_i > 0$, let $W_i = \{w_i^1, \ldots, w_i^{a_i}\}$, and if $a'_i > 0$, let $W'_i = \{w_i^{a_i+1}, \ldots, w_i^{c_i}\}$. Next, we define the preference list of every woman w, say P_w, as follows. For every $w \in W_i$, man t is in P_w if and only if the valuation of course x_i for the TA t is q_i. For every woman $w \in W'_i$, man t is in the preference list of w if and only if the valuation of course x_i for TA t is positive. Recall that in an instance of the Stable Matching problem a man m is on the preference list of a woman w if and only if w is in the preference list of m. Next, we define the ordering of men in the preference list of every woman, which is based on the grades. Since no two TAs have the same grade for a course, for men t, t' in P_w, woman w prefers t more than t' if and only if $g_t(w) > g_{t'}(w)$. Next, we define the ordering of women in the preference list of every man. Let P_t be the set of women in the preference list of the man t. Consider a man t. If $u_t(x_i) > u_t(x_j)$, where $x_i, x_j \in X$, then man t prefers women in $w \in W_i \cup W'_i$ more than woman $\hat{w} \in W_j \cup W'_j$, where $w, \hat{w} \in P_m$. Furthermore, if $w \in P_m \cap W_i$ and $\hat{w} \in P_m \cap W'_i$, then m prefers w more than $\hat{w}$. If $w, \hat{w} \in P_m \cap W_i$, then m order them arbitrarily; similarly women in $P_m \cap W'_i$ are ordered arbitrarily. This completes the construction of $\mathcal{J}$.

Proof of correctness and running time analysis can be found in the full version [24]. ◀

5 Parameterized (Approximation) Algorithms

In this section, we consider the problem in the realm of parameterized algorithms to cope with the intractability. We first consider the parameter m (the number of TAs.). The following result is due to trying all possible partitions of TAs.

▶ **Theorem 12 (♣).** *MEFE-Matching can be be solved in FPT(m), where m is the number of TAs.*

Next, we move our attention to the parameter n, the number of course. The next algorithm is similar to the one in Theorem 11. But, here we have constant number of distinct positive valuations for each course, rather than just two. So, we guess the number of TAs of

each value assigned to a course in a solution. Note that we also consider constant capacity for each course. The rest of the algorithm is similar. In the algorithm in Theorem 11, we know the upper bound for a_i, but here, we know the exact values, and hence add the edges accordingly in the Stable Matching instance.

▶ **Theorem 13.** *MEFE-Matching can be solved in FPT(n) when every course has constant number of distinct positive valuations and capacity, and for a course no two TAs have the same grade. Furthermore, each TA has distinct valuations for positively valued courses.*

Proof. Let $\mathcal{I}$ be an instance of MEFE-Matching that satisfies the constraints in the theorem statement. Let $v_i \colon T \to \{q_{1i}, \ldots, q_{r_i i}\}$, where r_i is a constant. Let a_{ji} be the number of TAs of value q_{ji} assigned to the course x_i in a solution, where $i \in [n], j \in [r_i]$. We guess the values of a_{ji}, for all $i \in [n], j \in [r_i]$, that satisfies the following two constraints.

$$\sum_{j=1}^{r_i} a_{ji} q_{ji} \geq k c_i \tag{3}$$

$$\sum_{j=1}^{r_i} a_{ji} = c_i \tag{4}$$

Note that we have $(c_i + 1)^{r_i}$ choices for vector $(a_{1i}, \ldots, a_{r_i i})$ for every course x_i. A vector is called *valid*, if it satisfies Equation 3 and 4. Since the capacity of each course and r_i is constant, we have constant choices for every course. Hence, we have c^n total choices, where c is a constant.

For each combination of n valid vectors (one for each course) $(a_{11}, \ldots a_{r_1 1}, \ldots, a_{1n}, \ldots a_{r_n n})$, we create an instance of the Stable Matching problem $\mathcal{J}$ as follows. The set of men is $M = T$, i.e., corresponding to every TA in T, we have a man in M. Corresponding to every course $x_i \in X$, we have a set of women W_{ji} that contains a_{ji} many women. If any of these sets are empty we ignore those sets. Let W_i be set of all women corresponding to course x_i. Due to the validity of the vector, corresponding to every course x_i, we have c_i women. Next, we define the preference list of every woman w, say P_w, as follows. For every $w \in W_{ji}$, man t is in P_w if the valuation of course x_i for the TA t is q_{ji}. Recall that in an instance the Stable Matching problem a man m is in the preference list of a woman w if and only if w is in the preference list of m. Thus, woman w is in P_m, the preference list of m, if and only if m is in P_w. Next, we define the ordering of men in the preference list of every woman, which is based on the grades. Let w be a women corresponding to the course x_i. Since no two TAs have the same grade for a course, for men t, t' in P_w, w prefers t more than t' if and only if $g_t(x_i) > g_{t'}(x_i)$. Next, we define the ordering of women in the preference list of every man. Consider a man t. If $u_t(x_i) > u_t(x_j)$, where $x_i, x_j \in X$, then man t prefers women in $w \in W_i$ more than woman in $\hat{w} \in W_j$, where $w, \hat{w} \in P_t$. If $w, w' \in P_t \cap W_i$, then we first note that there exists unique $j \in [r_i]$ such that $w, w' \in W_{ji}$, due to the construction. In this case, man t orders them arbitrarily. This completes the construction of $\mathcal{J}$. If any of the constructed instance of Stable Matching returns a women-saturating stable matching, then we return "yes", otherwise "no".

The proof of correctness can be found in the full version [24]. ◀

Next, we design a parameterized approximation scheme with respect to the parameter n, ϵ, and $\log v$, where v is maximum value assigned to a TA by a course. Our basic idea is that for appropriately chosen ϵ', we guess the number of TAs assigned to course x_i that have valuations in the range $[(1+\epsilon')^{j-1}, (1+\epsilon')^j)$, for every course $x_i \in X$. Then, we create the number of copies of a course accordingly and reduce to the Stable Matching problem as in Theorem 13.

▶ **Theorem 14 (♣).** *MEFE-Matching admits an algorithm that given a yes-instance outputs a matching in which for every course x, $\mathsf{AvgUtil}(x) \geq (1-\epsilon)k$, where $0 \leq \epsilon < 1$, when the capacity of each course is constant, no two TAs have the same grade for a course, and each TA assigns distinct utilities to each course, and runs in $\mathcal{O}(\mathsf{cap}^{\frac{n \log \mathsf{max_{val}}}{\log(1+\epsilon')}}(n+m)^{\mathcal{O}(1)})$, where cap is the maximum capacity of a course and $\mathsf{max_{val}}$ is the maximum valuation of a course for a TA, $\epsilon' = \frac{1}{1-\epsilon} - 1$.*

6 Relevance and Tractability of Results

In this section, we discuss the relevance of our results with respect to the constraints they require and their computational efficiency. We first contrast the results with a brute-force approach.

The brute-force algorithm is to try all possible allocations of TAs to faculty. Thus, the running time is $\mathcal{O}((n+1)^m)$ (+1 for the case that a TA might not be assigned to any course). Theorem 12 is this brute-force algorithm that leads to $\mathsf{FPT}(m)$ as $n \leq m$ for a yes-instance because every course has capacity at least one. Theorem 13 establishes a running time bound of $\mathcal{O}(c^n)$, where c is a constant. The number of TAs is usually larger than the number of courses; thus, this algorithm is faster than the brute-force algorithm. Note that the constraints in Theorem 13 do not change the running time of the brute-force algorithm. Similarly, the FPT-AS algorithm of Theorem 14 achieves a running time that improves upon the brute-force approach. We also believe that the constraints in Theorem 13 and 14 are natural. E.g., using TAs' course marks, which are likely distinct and can serve as grades. We can also ask TAs to submit distinct utilities for distinct courses. It is realistic to assume that a course's TA requirement remains constant (required in Theorems 13, 14) and that courses consistently split TAs into a constant number of utility brackets (e.g., good, average, bad) (required in Theorem 13).

We now move on to other results with practical constraints and polynomial running times. Theorem 11 considers bi-valued functions from the side of courses, capturing scenarios where instructors rank TAs as "like" or "dislike", a generalization of the binary case. For computationally hard problems, binary or bi-valued valuations are well-studied in Fair Allocation. Theorem 7, meanwhile, relies on the assumption that each course requires only one TA, which models contexts where only a few students opt for advanced electives and simplifies the assignment process.

7 Existence Results

In this section, we identify some yes and no instances of the problem.

7.1 Yes-instances of MEFE-Matching.

We begin by identifying instances where the solution is guaranteed to exist. Before presenting our first result, we define some notation.

Let $r_i \colon T \to [m]$ be a rank function such that t has $r_i(t)$th lowest grade in the course x_i, for each $i \in [n]$. Here, m denotes the size of the set T. Let $c = c_1 + \ldots + c_n$. We denote $k_i = g_t(x_i)$ where $t = r_i^{-1}(c)$. Note that if the number of TAs, m, is smaller than c, then $r_i^{-1}(c)$ will be undefined. However, we can easily identify such a case as a trivial non-instance since no feasible matching exists that meets each course's capacity constraint c_i.

Now, we are ready to present our first existential result.

▶ **Theorem 15 (♣).** *Let $\mathcal{I}$ be an instance of MEFE-Matching such that*

- **(i)** *every TA $t_i \in T$ has the following valuation function for courses, given by $u_i \colon X \to \{0, a\}$, for each $i \in [m]$;*
- **(ii)** *for every course $x \in X$ and TA $t \in T$, $v_x(t) = g_t(x)$, i.e., the utility of course x for TA t is the same as her grade in course x;*
- **(iii)** *for every subset $X' \subseteq X$, $|N(X')| \geq \sum_{x \in X'} c_x$.*
- **(iv)** *for every course x, there should not be two distinct TAs t, t' such that $u_t(x) = u_{t'}(x) = a$ but $g_t(x) = g_{t'}(x)$, i.e., no two TAs positively valuing a course should have the same grade for that course.*

Then, for $k = \min\{k_i \mid x_i \in X\}$, a MEFE matching always exists.

Our next positive existence result is because MEFE-Matching instance satisfying constraints in Theorem 16 can be reduced to Hospitals Residents problem (HR).

▶ **Theorem 16 (♣).** *Let $\mathcal{I}$ be an instance of MEFE-Matching such that all TAs and courses positively value each other and each TA must assign distinct valuation to courses and all the TAs have distinct grades/marks in a course. Then, MEFE always exist for $k = 1$.*

7.2 No-instances of MEFE-Matching.

Each yes-instance discussed in the previous section has multiple constraints. In this section, we discuss that none of the constraints mentioned for positive existence results are by themselves sufficient for the existence of a MEFE matching. If course utilities are grades, i.e., $v_j(t) = g_t(x_j)$ for all $x_j \in X, t \in T$, with no other constraints, the answer to the existence question is NO. For example, in an instance with a single course having capacity 1 and 2 TAs having the same grade, the unassigned TA will always envy the other, so no MEFE matching exists. If $k = 1$, with no other conditions, the answer is again NO, as the above example is a NO instance for this case. If all TAs and courses positively value each other, without further restrictions, the answer remains NO. For example, we take an instance with a single course having capacity 1 and 2 TAs, where both TAs positively value the course and the course positively values both TAs, and courses assign utilities less than k to both TAs. For this instance, no MEFE matching exists, since the course satisfaction criteria cannot be met. If TAs assign valuation functions having the range $0, a$ to courses given that a feasible matching exists, the answer to the existence question remains NO as the previous example is a NO instance.

8 Conclusion

In this paper, we considered many-to-one matching problem under two-sided preferences, and inspired from the TA allocation problem to the courses, we used welfare function on one side and envy-freeness type fairness criteria on the other side. Indeed, the application of problem is not limited to the TA allocation problem. We next list some of our specific open questions. Most of our algorithms require that "in a course, no two TAs have the same grade". Is this essential to derive those polynomial-time algorithms? In our $\mathsf{FPT}(n)$ algorithm, we require capacity and types of valuations of a course to be constant. What is the complexity with respect to n, when only one of them is constant?

References

1 Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A. Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artif. Intell.*, 322:103965, 2023. doi:10.1016/j.artint.2023.103965.

2 Keisuke Bando, Ryo Kawasaki, and Shigeo Muto. Two-sided matching with externalities: A survey. *Journal of the Operations Research Society of Japan*, 59(1):35–71, 2016.

3 Sayan Bandyapadhyay, Fedor V Fomin, Tanmay Inamdar, and Kirill Simonov. Proportionally fair matching with multiple groups. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 1–15, 2023. doi:10.1007/978-3-031-43380-1_1.

4 Robert Bredereck, Andrzej Kaczmarczyk, and Rolf Niedermeier. Envy-free allocations respecting social networks. In *AAMAS 2018*, pages 283–291, 2018. doi:10.1016/j.artint.2022.103664.

5 Xiaolin Bu, Zihao Li, Shengxin Liu, Jiaxin Song, and Biaoshuai Tao. Fair division with allocator's preference. In *International Conference on Web and Internet Economics*, pages 77–94, 2023. doi:10.1007/978-3-031-48974-7_5.

6 Jiehua Chen. Computational complexity of stable marriage and stable roommates and their variants. *CoRR*, abs/1904.08196, 2019. arXiv:1904.08196.

7 Sung-Ho Cho, Kei Kimura, Kiki Liu, Kwei-guu Liu, Zhengjie Liu, Zhaohong Sun, Kentaro Yahiro, and Makoto Yokoo. Fairness and efficiency trade-off in two-sided matching. In *AAMAS 2024*, pages 372–380, 2024. doi:10.5555/3635637.3662886.

8 M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer-Verlag, 2013.

10 Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 214–226. ACM, 2012. doi:10.1145/2090236.2090255.

11 Tomás Feder. *Stable Networks and Product Graphs*, volume 555. American Mathematical Soc., 1995.

12 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

13 Rupert Freeman, Evi Micha, and Nisarg Shah. Two-sided matching meets fair division. In Zhi-Hua Zhou, editor, *IJCAI 2021*, pages 203–209. ijcai.org, 2021. doi:10.24963/IJCAI.2021/29.

14 David Gale and Lloyd S Shapley. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

16 Sreenivas Gollapudi, Kostas Kollias, and Benjamin Plaut. Almost envy-free repeated matching in two-sided markets. In *WINE 2020*, pages 3–16, 2020. doi:10.1007/978-3-030-64946-3_1.

17 Anjali Gupta, Shreyans J Nagori, Abhijnan Chakraborty, Rohit Vaish, Sayan Ranu, Prajit Prashant Nadkarni, Narendra Varma Dasararaju, and Muthusamy Chelliah. Towards fair allocation in social commerce platforms. In *WWW 2023*, pages 3744–3754, 2023. doi:10.1145/3543507.3583398.

18 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Balanced Stable Marriage: How Close is Close Enough? *Theoretical Computer Science*, 883:19–43, 2021. doi:10.1016/j.tcs.2021.05.015.

19 Dan Gusfield and Robert W Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT press, 1989.

20 Ayumi Igarashi, Yasushi Kawase, Warut Suksompong, and Hanna Sumita. Fair division with two-sided preferences. In *IJCAI 2023*, pages 2756–2764, 2023. doi:10.1016/j.geb.2024.07.008.

21 Robert W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48(3):261–272, 1994. doi:10.1016/0166-218X(92)00179-P.

22 Robert W Irving, Paul Leather, and Dan Gusfield. An Efficient Algorithm for the "Optimal" Stable Marriage. *Journal of the ACM*, 34(3):532–543, 1987. doi:10.1145/28869.28871.

23 Robert W. Irving, David F. Manlove, and Gregg O'Malley. Stable marriage with ties and bounded length preference lists. *Journal of Discrete Algorithms*, 2009. doi:10.1016/j.jda.2008.09.003.

24 Pallavi Jain, Palash Jha, and Shubham Solanki. Fairness and efficiency in two-sided matching markets, 2025. doi:10.48550/arXiv.2504.10232.

25 Pallavi Jain and Rohit Vaish. Maximizing nash social welfare under two-sided preferences. In *AAAI 2024*, volume 38, pages 9798–9806, 2024. doi:10.1609/aaai.v38i9.28839.

26 Gili Karni, Guy N. Rothblum, and Gal Yona. On fairness and stability in two-sided matchings. In *ITCS 2022*, volume 215, pages 92:1–92:17, 2022. doi:10.4230/LIPIcs.ITCS.2022.92.

27 Michael P. Kim, Aleksandra Korolova, Guy N. Rothblum, and Gal Yona. Preference-informed fairness. In *ITCS 2020*, volume 151 of *LIPIcs*, pages 16:1–16:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.1145/3351095.3373155.

28 Donald Ervin Knuth. *Stable Marriage and its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*, volume 10. American Mathematical Soc., 1997.

29 Fuhito Kojima, Akihisa Tamura, and Makoto Yokoo. Designing matching mechanisms under constraints: An approach from discrete convex analysis. *Journal of Economic Theory*, 176:803–833, 2018. doi:10.1016/j.jet.2018.05.004.

30 Adam Kunysz. An algorithm for the maximum weight strongly stable matching problem. In *ISAAC*, volume 123 of *LIPIcs*, pages 42:1–42:13, 2018. doi:10.4230/LIPIcs.ISAAC.2018.42.

31 Shengxin Liu, Xinhang Lu, Mashbat Suzuki, and Toby Walsh. Mixed fair division: A survey. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *AAAI 2024*, pages 22641–22649. AAAI Press, 2024. doi:10.1609/aaai.v38i20.30274.

32 David F. Manlove. *Algorithmics of matching under preferences*. World Scientific, 2013. doi:10.1142/8591.

33 David G McVitie and Leslie B Wilson. The Stable Marriage Problem. *Communications of the ACM*, 14(7):486–490, 1971. doi:10.1145/362619.362631.

34 Shaily Mishra, Manisha Padala, and Sujit Gujar. Fair allocation of goods and chores - tutorial and survey of recent results. *CoRR*, abs/2307.10985, 2023. doi:10.48550/arXiv.2307.10985.

35 Shivika Narang, Arpita Biswas, and Yadati Narahari. On Achieving Leximin Fairness and Stability in Many-to-One Matchings. In *AAMAS 2022*, pages 1705–1707, 2022. doi:10.5555/3535850.3536083.

36 Gourab K Patro, Arpita Biswas, Niloy Ganguly, Krishna P Gummadi, and Abhijnan Chakraborty. Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. In *Proceedings of the web conference 2020*, pages 1194–1204, 2020. doi:10.1145/3366423.3380196.

37 Jay Sethuraman, Chung-Piaw Teo, and Liwen Qian. Many-To-One Stable Matching: Geometry and Fairness. *Mathematics of Operations Research*, 31(3):581–596, 2006. doi:10.1287/moor.1060.0207.

38 Chung-Piaw Teo and Jay Sethuraman. The Geometry of Fractional Stable Matchings and its Applications. *Mathematics of Operations Research*, 23(4):874–891, 1998. doi:10.1287/moor.23.4.874.

39 Yu Yokoi. Envy-free matchings with lower quotas. *Algorithmica*, 82(2):188–211, 2020. doi:10.1007/s00453-018-0493-7.

Improved Approximation for Pathwidth One Vertex Deletion and Parameterized Complexity of Its Variants

Satyabrata Jana
University of Warwick, UK

Soumen Mandal
Department of Mathematics, IIT Delhi, India

Ashutosh Rai
Department of Mathematics, IIT Delhi, India

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

Abstract

The pathwidth of a graph is a measure of how path-like the graph is. The Pathwidth One Vertex Deletion (POVD) problem asks whether, given an undirected graph G and an integer k, one can delete at most k vertices from G so that the remaining graph has pathwidth at most one. This is a natural variation of the classical Feedback vertex Set (FVS) problem, where the deletion of at most k vertices results in a graph of treewidth at most one. In this work, we investigate POVD in the realm of approximation algorithms. We first design a 3-approximation algorithm for POVD running in polynomial time. Then, using this constant factor approximation algorithm, we obtain a randomized parameterized approximation algorithm for POVD running in time $\mathcal{O}^*((h_\beta)^k)$, that improves the fastest existing running times for approximation ratios in the range $(1.76147, 3)$. Here the constant h_β depends on the approximation factor β alone and has value $2^{(3-\beta)}$, which lies in the range $(1, 2.3596)$, when $\beta \in (1.76147, 3)$.

Taking inspiration from two extensively studied problems, namely Connected FVS and Independent FVS, we investigate two variations of the POVD problem from the perspective of parameterized algorithms. These variations are the connected variant, called Connected pathwidth One Vertex Deletion (CPOVD) and the independent variant, called Independent Pathwidth One Vertex Deletion (IPOVD). While in CPOVD the subgraph $G[S]$ induced by the vertices to be deleted needs to be connected, in IPOVD it needs to be independent. Specifically, we show the following results.

- CPOVD can be solved in $\mathcal{O}^*(14^k)$ time and admits no polynomial kernel unless $\mathsf{NP} \subseteq \text{co-}\mathsf{NP}/\text{poly}$.
- IPOVD can be solved in $\mathcal{O}^*(7^k)$ time, and admits a kernel of size $\mathcal{O}(k^3)$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Pathwidth, Parameterized complexity, Approximation, Kernelization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.39

Funding *Satyabrata Jana*: Supported by the Engineering and Physical Sciences Research Council (EPSRC) via the project MULTIPROCESS (grant no. EP/V044621/1).
Soumen Mandal: Supported by the Council of Scientific and Industrial Research, India (file No. 09/0086(11535)/2021-EMR-I).
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).



© Satyabrata Jana, Soumen Mandal, Ashutosh Rai, and Saket Saurabh;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Treewidth is a fundamental graph parameter that measures how "tree-like" the graph is. The notion of treewidth was introduced by Robertson and Seymour in their seminal Graph Minors series [27]. Formally, it is defined via tree decompositions, where the goal is to map a graph into a tree-like structure of subsets of vertices of the graph, called *bags*, in a way that minimizes the size of the largest bag minus one. Treewidth plays a central role in the field of parameterized complexity and has been extensively studied due to its wide applicability in algorithm design, particularly for fixed-parameter tractable (FPT) algorithms. Many computationally hard problems become efficiently solvable when restricted to graphs of bounded treewidth.

A closely related parameter is *pathwidth*, which is a notion closely related to treewidth, and was also introduced by Robertson and Seymour in the Graph Minors series [26]. Here the underlying decomposition structure is restricted to paths instead of trees. While treewidth measures the tree-likeness of a graph, pathwidth measures how "path-like" the graph is.

The Feedback Vertex Set (FVS) problem is a classic graph modification problem where given a graph G and an integer k, the objective is to delete at most k vertices to obtain an acyclic graph (i.e., a forest). This problem can be viewed through the lens of treewidth, as forests are exactly the graphs with treewidth at most one. Due to this we can refer FVS problem as Treewidth One Vertex Deletion. FVS has been intensively studied in the FPT framework [12, 13, 16], leading to numerous algorithmic breakthroughs. Several variants of FVS have also been investigated. Two notable examples are: Connected Feedback Vertex Set (CFVS) [1, 6, 22] and Independent Feedback Vertex Set (IFVS)[2, 17, 21, 28], where in CFVS the subgraph of G induced by solution vertices S needs to be connected, and in IFVS it needs to be independent. Currently, the most efficient algorithm known for IFVS is provided by Li and Pilipczuk, which runs in $\mathcal{O}^*(3.619^k)$ time [17]. Regarding CFVS, Cygan et al.[7] gave the fastest known randomized algorithm for CFVS, achieving a runtime of $\mathcal{O}^*(4^k)$.

Many classical graph problems such as Vertex Cover, Feedback Vertex Set (FVS), Odd Cycle Transversal (OCT), and Dominating Set (DOM) have been extensively studied in the fields of parameterized complexity and graph algorithms. These problems are central to both theoretical research and practical applications, and their algorithmic properties – especially in the fixed-parameter tractability (FPT) setting. In recent years, increasing attention has been devoted to connected and independent variants of these problems, which often impose additional structural constraints and lead to new algorithmic challenges. Problems such as Connected Vertex Cover [14, 23], Connected DOM [11], Connected OCT [7] have emerged as natural extensions, where solutions are required to induce connected subgraphs. Similarly, Independent Dominating Set [18], Independent OCT [20], Independent s-t Separator [20], Independent Multicut [19], Independent Directed FVS [19], Independent Cutset [25] introduce independence constraints.

1.1 Pathwidth One Vertex Deletion

Analogous to FVS or Treewidth One Vertex Deletion, Philip et al.[24] introduced the Pathwidth-One Vertex Deletion (POVD) problem. In problem POVD, given a graph G and an integer k, the goal is to delete at most k vertices such that the remaining graph has pathwidth at most one. We define the problem formally as follows.

Pathwidth One Vertex Deletion (POVD)
Input: A graph $G = (V, E)$, a positive integer k.
Question: Does there exist $S \subseteq V(G)$ such that $|S| \leq k$ and pathwidth of $G - S$ is at most one?

It is known that a graph has pathwidth at most one if and only if it is a disjoint union of caterpillars [29]. A *caterpillar* is a special kind of tree where all vertices are within distance one of a central path called the *spine* [24, 29]. Hence, graphs of pathwidth at most one form a subclass of forests, exhibiting even simpler structure.

It follows from a general NP-hardness result of Lewis and Yannakakis [15] that this problem is NP-complete. Therefore, the POVD problem has attracted some attention in the parameterized complexity community and several FPT algorithms have been developed for this problem. Philip et al. [24] gave an algorithm for POVD with $\mathcal{O}^*(7^k)$ running time[1] and a kernel with $\mathcal{O}(k^4)$ vertices. Cygan et al. [8] improved these results by giving an algorithm with $\mathcal{O}^*(4.646^k)$ running time and a kernel with $\mathcal{O}(k^2)$ edges. The current fastest FPT algorithm for POVD has a running time of $\mathcal{O}^*(3.888^k)$[29]. Very recently, Esmer and Kulik [10] gave a parameterized approximation algorithm for POVD with a running time of $\mathcal{O}^*((3.888 - \varepsilon)^k)$ for some $\varepsilon > 0$, achieving approximation factors lying in the range $(1, 7)$.

1.2 Our Contributions

Following the line of research on FVS variants such as CFVS and IFVS, in this paper, we initiate the study of two natural extensions of POVD, one capturing independence, and the other emphasizing connectivity. Our research looks into these variants, revealing their intricacies with the following theme.

Connectivity vs **Independence**

Formally, we study the following connected and independent variants of POVD.

Connected Pathwidth One Vertex Deletion (CPOVD)
Input: A graph G, a positive integer k.
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G[S]$ is connected and pathwidth of $G - S$ is at most one?

Independent Pathwidth One Vertex Deletion (IPOVD)
Input: A graph G, a positive integer k.
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G[S]$ is independent and pathwidth of $G - S$ is at most one?

We obtain the following results for CPOVD and IPOVD.

- IPOVD admits a kernel of size $\mathcal{O}(k^3)$ and can be solvable in time $\mathcal{O}^*(7^k)$.
- CPOVD does not admit a polynomial kernel but can be solvable in time $\mathcal{O}^*(14^k)$.

In Section 4, we develop a kernel of size $\mathcal{O}(k^3)$ for IPOVD. Our approach builds on the framework of established kernelization algorithms for POVD [8, 24] and IFVS [21]. However, due to the inherited generality of our problem, we need to introduce substantial modifications to the reduction rules and perform a more detailed analysis. The size of our kernel matches, up to a constant factor, the best-known kernel size for IFVS. In contrast to IPOVD, in

[1] We use $\mathcal{O}^*$ notation to hide factors polynomial in the input size.

Section 5, we prove that CPOVD does not admit a polynomial kernel parameterized by the solution size unless NP $\subseteq$ co-NP/poly. This negative result follows via a straightforward reduction from the Connected Vertex Cover problem, which is also known not to have a polynomial kernel under the same assumption.

On the algorithmic side, in Section 6, we give an FPT algorithm for CPOVD running in time $\mathcal{O}^*(14^k)$. The core idea is to first eliminate certain obstructive structures from the graph through branching, simplifying the instance. Taking advantage of this simplified graph, we then invoke an FPT algorithm for Group Steiner Tree in a black-box fashion with carefully constructed input, allowing us to test whether any minimal POVD solution can be extended to a connected set that meets the size constraint. Beyond the $\mathcal{O}(k^3)$ kernel for IPOVD, in Section 7 we also present an FPT algorithm for IPOVD running in time $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. This algorithm is conceptually similar to the earlier FPT approach for CPOVD: we first branch to remove some of the obstructive structures, and then, again with the help of the simplified structure of the residual graph, check whether any minimal POVD solution of size at most k is independent.

Interestingly, despite the active study of approximation algorithms for other vertex deletion problems, no polynomial-time approximation algorithm better than the trivial 7-approximation was previously known for POVD. We provide a 3-approximation algorithm for POVD running in polynomial time. This algorithm is based on two key observations that any pathwidth-one vertex deletion set also forms a feedback vertex set and, POVD can be solved in polynomial time when the input graph is acyclic.

Furthermore, we investigate POVD in the realm of parameterized approximation. Recently, Esmer and Kulik [10], presented a parameterized approximation algorithm for POVD that achieve an approximation guarantee with factors in the range $(1, 7)$ in randomized FPT running time, that improves over the best known parameterized algorithm for POVD. In Section 3.3, with the help of the 3-approximation algorithm, we obtain another parameterized approximation algorithm that improves upon the existing randomized FPT running times of algorithms given by [10] for approximation ratios in the range $(1.76147, 3)$. Due to space constraints, some proofs have been moved to Appendix (Appendix A).

2 Preliminaries

All the graphs we consider are simple, undirected and unweighted graphs. For a graph G, we denote the set of vertices of the graph by $V(G)$ and the set of edges of the graph by $E(G)$. We denote $|V(G)|$ by n, where the graph is clear from the context. An edge with endpoint u and v is denoted by uv. For a set $S \subseteq V(G)$, the subgraph of G *induced* by S is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{uv \in E(G) : u, v \in S\}$. We denote the subgraph $G[V(G) \setminus S]$ as $G - S$. For any graph $G = (V, E)$ and $v \in V$, a vertex $u \in V$ is said to be a neighbour of v if $vu \in E(G)$. The open neighbourhood of v is denoted by $N(v)$ and it is defined as $N(v) = \{u \in V : vu \in E(G)\}$. We denote degree of a vertex v by $d(v)$ where $\deg(v) = |N(v)|$. The closed neighbourhood of a vertex v is $N[v] = N(v) \cup \{v\}$. By $\Delta(G)$, we denote the maximum degree of a vertex in G. We denote a polynomial function of n as $\text{poly}(n)$. A set $S \subseteq V$ is a *pathwidth-one vertex deletion set* (povd) if $G - S$ has pathwidth at most one. If $G[S]$ is an independent set, then S is an independent povd (ipovd). If $G[S]$ is connected, then S is a connected povd (cpovd).

Parameterized Complexity. We refer to [5] for an introduction to the area. A parameterized problem (x, k) where x is a string over Σ and a parameter $k \in \mathbb{N}$ is fixed-parameter tractable (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. A

kernelization algorithm, or in short, *kernelization*, for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(X, k) \in \Gamma^* \times \mathbb{N}$, outputs in time polynomial in $|X| + k$, a pair $(X', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(X, k) \in \Pi$ if and only if $(X', k') \in \Pi$ and (b) $|x'|, |k| \leq g(k)$, where g is some computable function depending only on k. The output of kernelization (X', k') is referred to as the kernel and the function g is referred to as the size of the kernel. If $g(k) \in k^{\mathcal{O}(1)}$, then we say that Π admits a polynomial kernel.

Branching Algorithm. We need to define the notion of a branching tree for the analysis of our algorithms. An algorithm or a procedure $\mathcal{A}$ on an instance I of a problem is called a branching algorithm if $\mathcal{A}$ can potentially recursively call itself on instances $I_1, I_2, \ldots, I_t$ for $t \geq 2$ such that output of $\mathcal{A}$ depends on the output of the recursive calls. A run of the algorithm on an instance I is said to have executed a branching step, if it recursively calls itself on at least two instances. A branching tree associated with an algorithm $\mathcal{A}$ and an input instance I is a tree where the root node of the tree is associated with the instance I. If on an instance I, branching step is executed and recursive call is made on the instances $I_1, I_2, \ldots, I_t$, then the node associated with I in the branching tree would have t children, each associated with one of the instances on which the recursive call was made. If there is no branching step executed on any of the instances associated with a node in the branching tree, then that node would be a leaf node in the branching tree.

Now, we state some useful known results on graphs with pathwidth at most one.

▶ **Fact 1** ([29, Lemma 1]). *Let G be a graph. The following assertions are equivalent.*

- *The pathwidth of G is at most 1.*
- *G is a caterpillar forest.*
- *G is acyclic and does not contain a subgraph isomorphic to T_2 (see Figure 1).*

▶ **Fact 2** ([24, Lemma 3]). *Let $\mathcal{S} = \{T_2, K_3, C_4\}$, where C_4 is a cycle of length 4. If G is a graph that does not contain any element of $\mathcal{S}$ as a subgraph, then each connected component of G is either a tree or a cycle with zero or more pendant vertices ("hairs") attached to it.*

3 Approximation for POVD

In this section, we describe the approximation algorithms for Pathwidth One Vertex Deletion. We first obtain a 3-approximation algorithm for POVD that runs in polynomial time. Then using this polynomial time approximation and results of [10], we obtain parameterized β-approximation for every $\beta \in [1, 3]$, the running time for which scales between the running times of the best known FPT algorithm for POVD ($\beta = 1$) and the polynomial time 3-approximation algorithm designed in Section 3.2 ($\beta = 3$). Before introducing these approximation techniques, we begin by proving a helpful result: POVD can be solved in polynomial time on acyclic graphs.

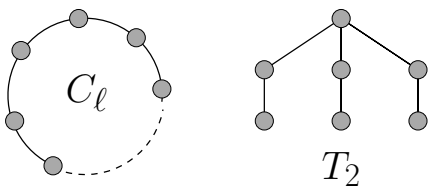


Figure 1 Forbidden subgraphs in graphs of pathwidth at most one. C_ℓ denotes a cycle of size ℓ.

3.1 Polynomial Time Algorithm Solving POVD on Acyclic Graphs

We first solve POVD on trees using a simple greedy approach. By Fact 1, the obstructions are cycles and the graph T_2. Since the input is a tree, the only possible obstruction is T_2, which has seven vertices, see Figure 1.

Given a tree H, we first root it at an arbitrary vertex $r \in V(H)$ using BFS. We then process the tree in a bottom-up manner to detect occurrences of T_2. Whenever a copy of T_2 is encountered for the first time in this traversal, we select a vertex v in the highest layer (closest to the root) of that copy. We include v in the solution set and remove v together with the entire subtree rooted at v from H. The process is then repeated: we again scan the tree bottom-up, locate the next copy of T_2, pick a highest-layered vertex v, and remove its subtree.

The algorithm terminates once the tree contains no copy of T_2. At that point, the set of selected vertices is returned as the solution.

We now argue the correctness of this approach.

▷ Claim 1. The above algorithm returns a minimum pathwidth-one vertex deletion set when the input graph is a tree.

Proof. Let H be a tree and let A be a copy of T_2 in H such that our algorithm identifies A during its first execution and selects one of the topmost vertices v from A into the solution. To prove correctness, it suffices to show that for any other T_2, say B in H intersecting A, the vertex v lies in $V(A) \cap V(B)$. If this holds, then choosing v is always as good as choosing any other vertex from A.

Assume for contradiction that $v \notin V(B)$, while $V(A) \cap V(B) \neq \emptyset$. Then there must exist a vertex $u \in V(A) \cap V(B)$ with $u \neq v$. Since v is chosen as one of the topmost vertices of A, the vertex u must occur at the same level or lower than v in the rooted tree. This would imply that there exist two distinct root-to-u paths in H: one passing through v and another through B that avoids v. Consequently, this would create a cycle in H, contradicting the assumption that H is a tree. Hence, v must belong to every T_2 intersecting A, establishing the claim. ◁

Since BFS runs in polynomial time and each scan for T_2 also takes polynomial time, the overall running time of the greedy algorithm is polynomial. Thus, POVD can be solved in polynomial time on trees. Moreover, since every connected component of an acyclic graph is itself a tree, we can apply the same approach independently on each component of an acyclic graph. This yields the following theorem.

▶ **Theorem 2.** POVD *can be solved in polynomial time on acyclic graphs.*

3.2 Polynomial Time 3-approximation for POVD

In this section, we prove the following result.

▶ **Theorem 3.** *There exists a polynomial time* 3*-approximation algorithm for* POVD.

The approach for our approximation algorithm is based on two key observations that any pathwidth-one vertex deletion set also forms a feedback vertex set and, as shown in the previous subsection, POVD can be solved in polynomial time when the input graph is acyclic. Our approximation algorithm proceeds as follows.

- Given a graph G, we first compute a 2-approximate feedback vertex set S_1, using known 2-approximation algorithms for the Feedback Vertex Set problem [3, 4]. Removing S_1 from G yields a graph $G - S_1$ that is acyclic.

- Next, we compute the minimum pathwidth-one vertex deletion set for $G - S_1$ using the polynomial-time algorithm described in Section 3.1. Let this set be S_2. Finally, we return the set $S = S_1 \cup S_2$.

Since S_2 is a pathwidth-one vertex deletion for $G - S_1$, $S = S_1 \cup S_2$ must be a pathwidth-one vertex deletion for G. Also, the above-described algorithm runs in polynomial time, as S_1 can be found in polynomial time using one of the 2-approximation algorithms for Feedback Vertex Set, and for S_2, we again need polynomial time only because of the result in Section 3.1. We next argue for the size bound of $S = S_1 \cup S_2$ in the following claim.

▷ Claim 4. For any given graph G, the size of S, returned by the algorithm described above, can be at most three times the size of an optimal pathwidth-one vertex deletion set of G.

Proof. Let S' be an optimal pathwidth-one vertex deletion for the graph G. We want to prove that $|S| \leq 3|S'|$. We know that $G - S'$ is a caterpillar forest. Also, let F be an optimal fvs for G. Since $G - S'$ is also acyclic, we must have $|F| \leq |S'|$. Again, since S_1 is a 2-approximate fvs solution, we get $|S_1| \leq 2|F| \leq 2|S'|$. On the other side, S_2 is an optimal pathwidth-one vertex deletion for $G - S_1$, which implies $|S_2| \leq |S'|$. Combining these, we get, $|S| = |S_1| + |S_2| \leq 2|S'| + |S'| \leq 3|S'|$. ◁

3.3 Parameterized Approximation for POVD

In this subsection, we improve the running time of the parameterized approximation algorithm for POVD introduced by Esmer and Kulik [10], for approximation factors between 1.76147 and 3. Recall that in [10], the authors provided running time for all approximation factors in the range from 1 to 7. However, due to our 3-approximation described in the previous subsection, it is no longer necessary to consider approximation factors greater than 3 in the parameterized setting for POVD. By combining our 3-approximation with the "Sampling with a black box" technique given in [10], we obtain faster β-approximation algorithms for all β such that $\beta \in (1.76147, 3]$.

Esmer and Kulik [10] showed that a β-approximation for a vertex deletion minimization problem π can be obtained in time $\mathcal{O}^*((\text{convert}(\alpha, \beta, c, q))^k)$ time, where the precondition is an α-approximation algorithm for the problem π that runs in time $\mathcal{O}^*(c^k)$ with $c \leq \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| q\right)\right)$, and that π has a sampling step where success probability of each step is at least q. Here $\mathcal{D}(\cdot\|\cdot)$ denotes Kullback-Leibler divergence[2]. The formula of $\text{convert}(\alpha, \beta, c, q)$ is as follows.

$$\text{convert}(\alpha, \beta, c, q) := \begin{cases} c \cdot \exp\left(\frac{\delta^*_{\text{right}} \cdot \mathcal{D}\left(\frac{1}{\delta^*_{\text{right}}} \| q\right) - \ln(c)}{\delta^*_{\text{right}} - \alpha} \cdot (\beta - \alpha)\right) & \text{if } \delta^*_{\text{right}}(\alpha, c, q) \geq \beta \geq \alpha; \\ c \cdot \exp\left(\frac{\delta^*_{\text{left}} \cdot \mathcal{D}\left(\frac{1}{\delta^*_{\text{left}}} \| q\right) - \ln(c)}{\delta^*_{\text{left}} - \alpha} \cdot (\beta - \alpha)\right) & \text{if } \delta^*_{\text{left}}(\alpha, c, q) \leq \beta \leq \alpha; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| q\right)\right) & \text{otherwise.} \end{cases} \tag{1}$$

[2] Kullback-Leibler Divergence: given two numbers $a, b \in [0, 1]$, the Kullback-Leibler divergence of a and b is defined to be

$$\mathcal{D}(a\|b) = a \cdot \ln\left(\frac{a}{b}\right) + (1 - a) \cdot \ln\left(\frac{1 - a}{1 - b}\right)$$

Here, $\delta^*_{\text{left}}(\alpha, c, q) \in (1, \alpha]$ and $\delta^*_{\text{right}}(\alpha, c, q) \in \left[\alpha, \frac{1}{q}\right]$ are defined to be the unique numbers which satisfy

$$\mathcal{D}\left(\frac{1}{\alpha} \| \frac{1}{\delta^*_{\text{left}}(\alpha, c, q)}\right) = \mathcal{D}\left(\frac{1}{\alpha} \| \frac{1}{\delta^*_{\text{right}}(\alpha, c, q)}\right) = \mathcal{D}\left(\frac{1}{\alpha} \| q\right) - \frac{\ln(c)}{\alpha}. \tag{2}$$

Note that for POVD, it was previously known that an α-approximation with $\alpha = 1$ (which is same as an exact algorithm) can be obtained in time $\mathcal{O}^*(3.888^k)$ [29], and there is also a sampling step for POVD that succeeds with probability $q = \frac{1}{7}$ [10]. Building on this precondition, a β-approximation has been developed in [10], which runs in time $\mathcal{O}^*\left((d_\beta)^k\right)$ for every $1 \leq \beta \leq 7$, where

$$d_\beta = \text{convert}(1, \beta, 3.888, \frac{1}{7}) = \begin{cases} 3.888 \cdot (0.519)^{\beta-1} & \text{if } 1 \leq \beta \leq 1.8; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| \frac{1}{7}\right)\right) & \text{if } 1.8 < \beta \leq 7. \end{cases} \tag{3}$$

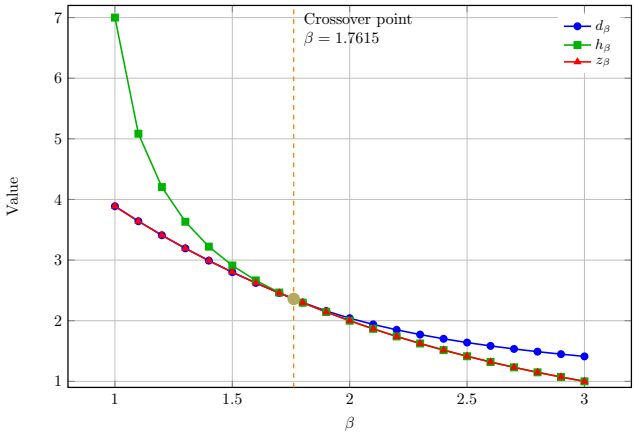


Figure 2 Comparison between the bases of the running times: d_β corresponding to the β-approximation from [10], and z_β corresponding to the β-approximation presented in Theorem 5.

Now, because of our new α-approximation with $\alpha = 3$ that runs in time $\mathcal{O}^*(c^k)$ with $c = 1$ (since it is a polynomial time 3-approximation), from Equation (1), we can obtain another parameterized β-approximation for every $\beta \in [1, 3]$, running in time $\mathcal{O}^*\left((h_\beta)^k\right)$, where

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right).$$

Now, we compute convert $\left(3, \beta, 1, \frac{1}{7}\right)$. Note that, to evaluate convert $\left(3, \beta, 1, \frac{1}{7}\right)$ for $\beta \leq 3$, we need the value of $\delta^*_{\text{left}}(3, 1, \frac{1}{7})$. Let $x = \delta^*_{\text{left}}(3, 1, \frac{1}{7})$, then from Equation (2), we have

$$\mathcal{D}\left(\frac{1}{3} \| \frac{1}{x}\right) = \mathcal{D}\left(\frac{1}{3} \| \frac{1}{7}\right) - \frac{\ln(1)}{3}.$$

This implies,

$$\frac{1}{3} \cdot \ln\left(\frac{x}{3}\right) + \frac{2}{3} \cdot \ln\left(\frac{2x}{3(x-1)}\right) = \frac{1}{3} \cdot \ln\left(\frac{7}{3}\right) + \frac{2}{3} \cdot \ln\left(\frac{14}{18}\right).$$

Further simplifying, we get

$$4x^3 = \frac{7^3 \cdot (x-1)^2}{9} \implies x = \frac{7}{4}, \text{ as } x = \delta^*_{\text{left}}(3, 1, \frac{1}{7}) \in (1, 3].$$

Now, using $\delta^*_{\text{left}}(3, 1, \frac{1}{7}) = \frac{7}{4}$ in Equation (1), we get

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right) = \begin{cases} \exp\left(\frac{\frac{7}{4}\cdot\mathcal{D}\left(\frac{1}{\frac{7}{4}} \| \frac{1}{7}\right)}{\frac{7}{4}-3} \cdot (\beta - 3)\right) & \text{if } 1.75 \leq \beta \leq 3; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| \frac{1}{7}\right)\right) & \text{otherwise.} \end{cases}$$

This gives

$$h_\beta = \text{convert}\left(3, \beta, 1, \frac{1}{7}\right) = \begin{cases} 2^{(3-\beta)} & \text{if } 1.75 \leq \beta \leq 3; \\ \exp\left(\beta \cdot \mathcal{D}\left(\frac{1}{\beta} \| \frac{1}{7}\right)\right) & \text{otherwise.} \end{cases} \tag{4}$$

Thus, we now have two different parameterized β-approximation algorithms for POVD, running in time $\mathcal{O}^*((d_\beta)^k)$ and $\mathcal{O}^*((h_\beta)^k)$, respectively. By comparing d_β and h_β, we observe that $d_\beta < h_\beta$ when $1 \leq \beta < 1.76147$, while $h_\beta < d_\beta$ when $1.76147 < \beta \leq 3$.

Combining these observations, we obtain the following theorem.

▶ **Theorem 5.** *For each* $1 \leq \beta \leq 3$, POVD *has a* β*-approximation algorithm running in time* $\mathcal{O}^*((z_\beta)^k)$*, where*

$$z_\beta = \begin{cases} 3.888 \cdot (0.519)^{(\beta-1)} & \text{if } 1 \leq \beta \leq 1.76147; \\ 2^{(3-\beta)} & \text{if } 1.76147 < \beta \leq 3. \end{cases}$$

4 Kernelization for Independent POVD

In this section, we present a polynomial kernel of size $\mathcal{O}(k^3)$ for IPOVD. Given an instance (G, k), our kernelization procedure first reduces it to an equivalent instance of a more general problem called Ext-IPOVD. In Ext-IPOVD, the input is a triple (G, R, k), and the goal is to determine whether there exists an independent pathwidth-one vertex deletion set in G that is disjoint from the set R. For any instance (G, k) of IPOVD, we can create an equivalent instance of Ext-IPOVD by simply setting $R = \emptyset$. Note that when $k \leq 3$, we can solve an input instance (G, k) of IPOVD in polynomial time by enumerating all possible subsets of size at most k. Thus, for a given instance (G, k), we assume that $k > 3$.

Table 1 Values of d_β (from [10]), h_β (from Equation (4)) and z_β (from Theorem 5) for β ranging from 1.0 to 3.0 in increments of 0.1. Highlighted row shows the point where z_β starts outperforming d_β.

β	d_β	h_β	z_β	β	d_β	h_β	z_β
1.0	3.888	7.0000	3.888	2.0	2.0417	2.0000	2.0000
1.1	3.6412	5.0846	3.6412	2.1	1.9391	1.8660	1.8660
1.2	3.4100	4.2041	3.4100	2.2	1.8498	1.7411	1.7411
1.3	3.1936	3.6324	3.1936	2.3	1.7713	1.6245	1.6245
1.4	2.9908	3.2220	2.9908	2.4	1.7018	1.5157	1.5157
1.5	2.8010	2.9102	2.8010	2.5	1.6399	1.4142	1.4142
1.6	2.6232	2.6642	2.6232	2.6	1.5844	1.3195	1.3195
1.7	2.4567	2.4647	2.4567	2.7	1.5346	1.2311	1.2311
1.7615	2.3596	2.3596	2.3596	2.8	1.4896	1.1486	1.1486
1.8	2.3007	2.2973	2.2973	2.9	1.4487	1.0717	1.0717
1.9	2.1604	2.1435	2.1435	3.0	1.4115	1.0000	1.0000

We then design a kernel of size $\mathcal{O}(k^3)$ for Ext-IPOVD. Once we obtain that we then convert it back into an equivalent instance of IPOVD of size $\mathcal{O}(k^3)$. The following lemma formalizes this transformation.

▶ **Lemma 6.** *Given an instance (G, R, k) of size $\mathcal{O}(k^3)$ for* Ext-IPOVD*, we can construct an equivalent instance $(H, k+1)$ for* IPOVD *of size $\mathcal{O}(k^3)$.*

Proof. Let (G, R, k) be an instance of Ext-IPOVD with size $\mathcal{O}(k^3)$. From (G, R, k), we create an equivalent instance $(H, k+1)$ of IPOVD as follows. We define the vertex set $V(H)$ by extending $V(G)$ with $2k+5$ additional vertices $\{u\} \cup \{x_1, \ldots, x_{k+2}\} \cup \{y_1, \ldots, y_{k+2}\}$. The edge set $E(H)$ is formed the following way. We extend the edge set $E(H)$ by adding the the following edges. For each $i \in [k+2]$, we add the edges $\{x_iy_i, ux_i, uy_i\}$, and then add an edge ur for every $r \in R$.

Observe that any ipovd solution Q for H of size at most $k+1$ must include the vertex u. Otherwise, Q would need to include at least one vertex from each of the $k+2$ disjoint pairs $\{x_i, y_i\}$, contradicting the size bound of $k+1$. Since Q is an independent set that includes u, it cannot contain any vertex from R. Therefore, $Q \setminus \{u\}$ forms an independent set in G of size at most k, disjoint from R. Furthermore, as G is an induced subgraph of $H \setminus \{u\}$, $Q \setminus \{u\}$ is also a valid povd solution for G. Conversely, let Q' be an ipovd solution of size at most k for the instance (G, R, k). Since Q' avoids all vertices in R, the set $Q' \cup \{u\}$ forms a valid ipovd solution for H of size at most $k+1$. Finally, since the size of G is bounded by $\mathcal{O}(k^3)$ and we only introduce $\mathcal{O}(k)$ new vertices and $\mathcal{O}(k^3)$ additional edges, the total size of H remains bounded by $\mathcal{O}(k^3)$. ◀

So now we turn our attention to kernelization for Ext-IPOVD. We develop several reduction rules for this purpose and apply them exhaustively to any given instance (G, R, k) of Ext-IPOVD. When multiple reduction rules are applicable simultaneously, we always choose the one with the smallest index. Once no further reduction rules apply, we show that the size of the resulting instance is bounded by $\mathcal{O}(k^3)$.

Note that certain reduction rules can produce trivial `Yes` instances (the instance $(G = K_2, R = \emptyset, k = 0)$) or trivial `No` instances (the instance $(G = K_3, R = \emptyset, k = 0)$) for Ext-IPOVD. Some of our rules are slight adaptations of known reductions, while others are specifically tailored to this problem. Below, we present these rules and provide proofs of correctness for all non-trivial rules. We begin by describing three simple reduction rules.

▶ **Reduction Rule 1.** *If $k \leq 0$ and G has pathwidth at least two, return a trivial `No` instance.*

▶ **Reduction Rule 2.** *If $k \geq 0$ and G has pathwidth at most one, return a trivial `Yes` instance.*

▶ **Reduction Rule 3.** *If G has a connected component C such that C is caterpillar, remove C from the graph. Return the instance $(G - V(C), R \setminus V(C), k)$.*

The proof of the correctness of the above three reduction rules is straightforward. Below we present few other rules (Reduction Rules 4-8), correctness for each reduction rule, are provided in the Appendix (Section A.1).

▶ **Reduction Rule 4.** *Given an instance (G, R, k), if a vertex $v \in V(G)$ has more than $k+1$ pendant neighbors, retain any $k+1$ of them, ensuring that at least one is from R if v has any pendant neighbor in R. Let D_v be the set of deleted pendant neighbors, G' the resulting graph, and update $R' = R \setminus D_v$. Return the instance (G', R', k).*

Now, we will state two known reduction rules given in [24] with slight modification.

▶ **Reduction Rule 5.** *Let u be a vertex in G such that there exists a matching M of size $k+3$ in the graph $G-u$, where each edge in M has at least one endpoint in the neighborhood $N(u)$. If $u \in R$, then return trivial* `No` *instance. Otherwise, remove u from the graph, add all vertices in $N(u)$ to the set R, and decrease k by one. Return the instance $(G-u,\ R \cup N(u),\ k-1)$.*

▶ **Reduction Rule 6.** *Let $v_0, v_1, v_2, \ldots, v_{p+1}$ be a path in G such that, for every vertex v_i with $1 \leq i \leq p$, all its neighbors other than v_{i-1} and v_{i+1} are pendant vertices. If $p \geq 7$, contract the edge (v_3, v_4) in G, and assign the resulting vertex v_3^4 in R if both v_3 and v_4 are in R. Denote the resulting graph by G'. The new instance is $\big(G',\ R',\ k\big)$, where*

$$R' = \begin{cases} \big(R \setminus \{v_3, v_4\}\big) \cup \{v_3^4\}, & \text{if } v_3^4 \in R, \\ R \setminus \{v_3, v_4\}, & \text{otherwise.} \end{cases}$$

Towards proving the correctness of Reduction Rule 6, we use the following observation.

▶ **Observation 7** ([24, Observation 2])**.**

1. *Let G be a graph that contains at least one cycle. Any graph G' obtained from G by contracting an edge of G also contains at least one cycle (possibly with parallel edges).*
2. *Let G be a graph that contains a T_2 as a subgraph. If G' is a graph obtained from G by contracting an edge uv where either u or v (or both) is not part of any T_2 in G, then G' also contains a T_2 as a subgraph.*

Let v be a vertex in a graph G, and let $\ell \in \mathbb{N}$. An *ℓ-flower* passing through v is a set of ℓ distinct cycles in G such that each cycle contains v and no two cycles share any vertex other than v. The vertex v is said to be at the center of the flower.

▶ **Reduction Rule 7.** *Let u be a vertex of the graph G such that u is the center of a $(k+1)$-flower. If $u \in R$, then return trivial* `No` *instance. Otherwise, remove u from the graph, add all vertices in $N(u)$ to the set R, and decrease k by one. Return the instance $\big(G[V(G) \setminus \{u\}],\ R \cup N(u),\ k-1\big)$.*

▶ **Lemma 8.** *Let $\alpha = \max\{5, k+2\}$. Consider an* Ext-IPOVD *instance (G, R, k). Assume that there is a vertex $u \in V(G)$ of degree at least $(4k+7)+\alpha(k+2)$ and Reduction Rules 1-7 cannot be applied. Then one can, in polynomial time, find disjoint sets of vertices $X', Y' \subseteq V(G) \setminus \{u\}$ and a function $\phi : X' \to \binom{Y'}{\alpha}$ satisfying the following properties:*

1. *every vertex in Y' is connected by a single edge to u,*
2. *Y' is an independent set in G,*
3. *$N(Y') = X' \cup \{u\}$, and each vertex from Y' has a neighbor in X',*
4. *the sets $\phi(x) \subseteq N(x)$ are disjoint for any two different $x \in X'$ (as before we denote the elements of $\phi(x)$ by $y_1^x, y_2^x, \ldots, y_\alpha^x$ in Y' and call them the private neighbors of x).*

The above mentioned lemma is similar to Lemma 7 in [8]. The only difference lies in the bound on $\deg(u)$. Since our graph is simple, the maximum number of non-pendant neighbors of u can be at most $k+1$, whereas in their case it is $2k+1$. The proof of our Lemma 8 follows directly from the proof of Lemma 7 in [8].

▶ **Reduction Rule 8.** *Let u be a vertex of degree at least $(4k+7)+\alpha(k+2)$, and assume that none of the earlier reduction rules are applicable. Let X' and Y' be the nonempty sets obtained by applying Lemma 8. Construct the graph G' by deleting all but two edges between u and the vertices y_i^x, for each $x \in X'$. Then, put all vertices in Y' to R. Return the instance $\big(G',\ R \cup Y',\ k\big)$.*

If the kernelization algorithm for Ext-IPOVD returns a trivial `Yes` or trivial `No` instance, it is straightforward to observe that the size of such instances is constant. Therefore, we now focus on the case where the kernelization produces a non-trivial instance (G', R', k'). Assuming the original input instance (G, R, k) is a `Yes`-instance, we will argue that the size of G' is bounded by $\mathcal{O}(k^3)$. By correctness of the reduction rules, we have that (G, R, k) is a `Yes`-instance if and only if (G', R', k') is a `Yes`-instance. Assume that (G', R', k') is a `Yes`-instance and let S be a solution of size at most k'. Since no reduction rule increases the value of k, it follows that $k' \leq k$. We now partition the set $V^\star = V(G') \setminus S$ into the following subsets.

$$V_1 = \Big\{v \in V^\star : \deg_{G'}(v) = 1, N(v) \cap S \neq \emptyset\Big\}; \quad V_2 = \Big\{v \in V^\star : \deg_{G'}(v) = 1, N(v) \cap S = \emptyset\Big\};$$

$$V_3 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 1, N(v) \cap S \neq \emptyset\Big\}; \; V_4 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, N(v) \cap S \neq \emptyset\Big\};$$

$$V_5 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, N(v) \cap S = \emptyset, \exists u \in N(v) : u \in V_3\Big\};$$

$$V_6 = \Big\{v \in V^\star : \deg_{G'-S}(v) > 2, v \notin V_4 \cup V_5\Big\}; \; V_7 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 2, N(v) \cap S \neq \emptyset\Big\};$$

$$V_8 = \Big\{v \in V^\star : \deg_{G'-S}(v) = 2, N(v) \cap S = \emptyset\Big\}.$$

▶ **Lemma 9.** *If the input instance of* Ext-IPOVD *is* `Yes` *instance, then* $|V(G')| \leq 61k^3 + 319k^2 + 414k + 42$*, and* $|E(G')| \leq 62k^3 + 327.5k^2 + 423.5k + 42$.

Proof. We assume that none of the reduction rules are applicable to the instance (G', R', k'). We now proceed to bound the size of each V_i individually.

Vertices in V_1 have exactly one neighbor, which must lie in S. Since $|S| \leq k$ and, by Reduction Rule 4, each vertex in S can have at most $k + 1$ such pendant neighbors, the size of V_1 is bounded by $k(k + 1) = k^2 + k$.

Next, observe that if $v \in V_7$ or $v \in V_4$, then v must lie on the spine of a caterpillar in the subgraph $G' - S$. Suppose there are i vertices from V_7 in a caterpillar. Then, in the subgraph induced by these i vertices and their neighbors, there exists a matching of size at least $\lfloor \frac{i}{2} \rfloor$. Now consider a vertex $u \in S$ that is adjacent to more than $3k + 9$ vertices of V_7. In the graph induced by these neighbors and their adjacent vertices, we can obtain a matching of size at least $k + 3$, where each matching edge has at least one endpoint adjacent to u. This would trigger Reduction Rule 5, contradicting our assumption. Therefore, each vertex in S can be adjacent to at most $3k + 9$ vertices of V_7. By the same reasoning, each vertex in S can also be adjacent to at most $3k + 9$ vertices of V_4. Hence, when Reduction Rule 5 is not applicable, the total number of vertices in each of V_7 and V_4 is bounded by $k(3k + 9) = 3k^2 + 9k$.

Notice that, for every $v \in V_5$, there is a private vertex $u \in V_3$ such that uv is an edge. Thus, if there are x such vertices $u \in V_3$ adjacent to vertices in V_5, then $|V_5| = x$. Moreover, in the subgraph of G' induced by these u and their neighbors in $V(G') \setminus S$, there exists a matching of size x. Consequently, if there is a vertex $w \in S$ adjacent to $k + 3$ such vertices $u \in V_3$ with neighbors in V_5, then Reduction Rule 5 would be applicable. Therefore, for each vertex in S, there can be at most $k + 2$ such vertices u, implying that $|V_5| \leq k(k + 2)$.

Observe that the vertices of V_6 and V_8 lie on the spine of a caterpillar in $G' - S$, such that neither they nor their pendant neighbors are adjacent to S. Since Reduction Rule 6 is no longer applicable, in any spine of a caterpillar in $G' - S$, there cannot be more than nine consecutive vertices from $V_6 \cup V_8$. Additionally, because Reduction Rule 6 is no longer applicable, there must be at least one vertex from V_4, V_5, or V_7 after any sequence of at most nine consecutive vertices from $V_6 \cup V_8$ in such a spine. Therefore, $|V_6 \cup V_8| \leq 9(7k^2 + 20k)$.

Note that the neighbors of any vertex in V_2 belong to some V_i with $i \geq 3$. Let $v \in V_2$ be adjacent to some $u \in V_3$. As before, there can be at most $k+2$ such vertices u (otherwise Reduction Rule 5 would be applicable). For any vertex in V_i with $i \geq 4$, there can be at most $k+1$ adjacent vertices in V_2 by Reduction Rule 4. Therefore, in total, the size of V_2 is bounded by $|V_2| \leq (k+2)+(k+1)\big(|V_4|+|V_5|+|V_6|+|V_7|+|V_8|\big) = 70k^3+270k^2+201k+2$.

Furthermore, if none of the reduction rules are applicable, then for any $u \in V(G')$, we have: $\deg(u) \leq (4k+7)+\alpha(k+2)$, where $\alpha = \max\{5, k+2\}$. Therefore, the total number of vertices adjacent to S is bounded by $(4k^2+7k)+\alpha(k^2+2k)$. Hence, $|V_3| \leq (4k^2+7k)+\alpha(k^2+2k) = k^3+8k^2+11k$ (note that according to our initial assumption, $\alpha = k+2$).

Hence, finally we get

$$|V(G')| = |S| + \sum_{i=1}^{8} |V_i| \leq 71k^3+349k^2+254k+2.$$

Similarly, the number of edges with one endpoint in S and the other in $V(G') \setminus S$ is at most $(4k^2+7k)+\alpha(k^2+2k) = (4k^2+7k)+(k+2)(k^2+2k)$. Since $G'-S$ is a caterpillar forest, the total number of edges in $G'-S$ is at most $|V(G') \setminus S|-1$. Finally, the number of edges in $G'[S]$ is bounded by $\frac{k(k-1)}{2}$. Therefore, the total number of edges in G' is at most $\Big(|V(G') \setminus S|-1\Big) + \Big[\text{number of edges between } S \text{ and } V(G') \setminus S\Big] + \frac{k(k-1)}{2} = (71k^3+349k^2+253k+1)+(k^3+8k^2+11k)+\frac{k(k-1)}{2} = 72k^3+357.5k^2+263.5k+1$. ◀

Now we are ready to give our final reduction rule, the correctness of which follows from the above lemma.

▶ **Reduction Rule 9.** *Let (G', R', k') be the reduced instance after applying Reduction Rule 1-8 exhaustively. If $|V(G')| > 71k^3+349k^2+254k+2$, or $|E(G')| > 72k^3+357.5k^2+263.5k+1$, then return trivial* `No` *instance.*

It is easy to verify that every reduction rule can be executed within polynomial time. Therefore, we have a kernelization algorithm for Ext-IPOVD, that either correctly returns a trivial `Yes` instance or a trivial `No` instance, or produces an equivalent instance (G', R', k') of size $\mathcal{O}(k^3)$. On the reduced Ext-IPOVD instance, we apply Lemma 6 to construct an equivalent IPOVD instance $(H, k'+1)$ whose size is also bounded by $\mathcal{O}(k^3)$.

Now, we can conclude the kernelization algorithm with the following theorem.

▶ **Theorem 10.** IPOVD *admits a kernel of size $\mathcal{O}(k^3)$.*

5 Kernel Lower Bound for CPOVD

In this section, we show that CPOVD does not admit a polynomial kernel when parameterized by the solution size k, unless NP $\subseteq$ co-NP/poly. We establish this by providing a parameter-preserving reduction from the Connected Vertex Cover problem to CPOVD.

▶ **Theorem 11.** CPOVD *does not admit polynomial kernel unless* NP $\subseteq$ *co-*NP*/poly.*

Proof. We give a straightforward parameter-preserving reduction from Connected Vertex Cover to CPOVD. In Connected Vertex Cover, we are given an undirected graph G along with an integer k and our goal is to check whether there is a subset $X \subseteq V(G)$ of size at most k such that $G[S]$ is connected and X contains at least one end-point of each edge. It is kwown that Connected Vertex Cover does not admit a polynomial kernel unless NP $\subseteq$ co-NP/poly [9].

For an instance $(G = (V, E), k)$ of Connected Vertex Cover, we construct an instance $(G' = (V', E'), k')$ of CPOVD as follows: $V' = V(G) \cup \{z_{uv} : uv \in E(G)\}$, and $E' = E(G) \cup \left\{e^u_{uv} = uz_{uv}, e^v_{uv} = vz_{uv} : uv \in E(G)\right\}$, and we set $k' = k$. We claim that G has a connected vertex cover of size at most k if and only if G' has a connected pathwidth-one vertex deletion set of size at most k.

In the forward direction, let S is a solution to Connected Vertex Cover. Now consider the graph $G' - S$. As S is vertex cover, in $G' - S$, the degree of every vertex z_{uv} is at most one, because at least one of u or v belongs to S. Now as S is a vertex cover of G, the graph $G' - S$ is a disjoint union of stars. Clearly, $G'[S]$ is connected since $G[S]$ is connected. Hence S is a connected pathwidth-one vertex deletion set of size at most k in G'.

In the backward direction, let G' has a connected pathwidth-one vertex deletion set S' of size at most k. We show that the vertex set $S = S' \setminus \{z_{uv} \mid uv \in E(G)\}$ is a connected vertex cover of G of size at most k. By the construction of G', there is a cycle $C_{uv} = \{u, z_{uv}, v\}$ for every edge $uv \in E(G)$. Therefore, by Fact 1, every POVD solution must include at least one vertex from each C_{uv}. In other words, S' contains at least one vertex from C_{uv} for every edge $uv \in E(G)$. Suppose S' contains z_{uv} for some edge $uv \in E(G)$. Since $G'[S']$ is connected and z_{uv} is adjacent only to u and v in G', S' must also include at least one of u or v. On the other hand, if S' does not contain z_{uv} for some edge uv, then it must contain at least one of u or v to cover the cycle. Thus, in all cases, S' contains at least one endpoint of every edge in G, implying that $S = S' \setminus \{z_{uv} \mid uv \in E(G)\}$ is a vertex cover of G of size at most k. Finally, removing the z_{uv} vertices from S' does not disconnect the induced subgraph in G' or in G, since each z_{uv} is either a leaf or part of a cycle in $G'[S']$. Therefore, S is a connected vertex cover of G of size at most k. This completes the proof. ◀

6 FPT Algorithm for CPOVD

In this section, we obtain the following result for CPOVD.

▶ **Theorem 12.** CPOVD *can be solved in* $14^k \cdot n^{\mathcal{O}(1)}$ *time.*

Overview. By Fact 2, if a graph G does not contain T_2, K_3, or C_4 as a subgraph, then each connected component of G is either a tree or a cycle with zero or more hairs. To compute a set of vertices whose removal makes the graph free of T_2, K_3, and C_4 as subgraphs, we can simply identify such a structure in the graph by brute-force search (in n^7 time) and branch on it. This results in at most 7-way branching, leading to at most 7^k different possible paths from the root to the leaves in the branching tree. Also, notice that any solution to POVD as well as CPOVD must contain all the vertices of one such path as a subset. In this algorithm, we first branch on the forbidden structures (T_2, K_3, and C_4), and then we call the algorithm of Group Steiner Tree at most 7^k times (once for each branching path).

Before going into the details of the algorithm, let us see Group Steiner Tree problem.

Group Steiner Tree
Input: An undirected graph G, vertex-disjoint subsets $S_1, \ldots, S_\ell \subseteq V$, and $p \in \mathbb{N}$.
Question: Does G contain a tree on at most p vertices that intersects each S_i, $i \in [\ell]$?

The following result is known for the Group Steiner Tree problem.

▶ **Theorem 13.** [22, Lemma 3] Group Steiner Tree *can be solved in* $2^\ell \cdot n^{\mathcal{O}(1)}$ *time.*

Now we describe our algorithm for CPOVD in detail. For a given instance (G, k) of CPOVD, we first check whether G contains a T_2, K_3, or C_4 as a subgraph (by exhaustive search) and branch on such a structure if it exists. In each branching step, the parameter k is decreased

by one. And update the graph to $G' = G - v$ and parameter to $k' = k - 1$. We stop branching on an instance (G', k') when either $k' \leq 0$ or G' contains none of T_2, K_3, or C_4 as a subgraph. Since T_2 contains 7 vertices, each branching step generate at most 7 sub-instances, leading to a 7-way branching. As k decreases by one at each step, the height of the branching tree is at most $k + 1$, and the total number of root-to-leaf paths is bounded by 7^k.

Let $\mathcal{P}$ denote the collection of vertex sets corresponding to each branching path, where each set $P_i \in \mathcal{P}$ consists of at most k vertices removed along that path. That is, if $P_i \in \mathcal{P}$, then there exists a root-to-leaf path in the branching tree such that P_i is the set of deleted vertices, and $|P_i| \leq k$. Clearly, $|\mathcal{P}| \leq 7^k$, and for each $P_i \in \mathcal{P}$, every connected component of $G - P_i$ is either a tree or a cycle with zero or more hairs. Now for each $P_i \in \mathcal{P}$, we check whether there is a connected povd solution of size at most k containing the vertex set P_i. Let $V(P_i) = \{v_{i_1}, v_{i_2}, \ldots, v_{i_{|P_i|}}\}$. Also Let $C_1^i, C_2^i, \ldots, C_{p_i}^i$ be the vertex sets corresponding to the cycle components in $G - P_i$. If $|P_i| + p_i > k$ then we immediately conclude that no such solution exists that contains the vertex set P_i. Else, i.e., when $|P_i| + p_i \leq k$, we invoke the Group Steiner Tree problem on the input graph G with the vertex-disjoint subsets $v_{i_1}, v_{i_2}, \ldots, v_{i_{|P_i|}}, C_1^i, C_2^i, \ldots, C_{p_i}^i$ with the budget parameter k. If Group Steiner Tree calls returns `Yes`, we return `Yes` for the input instance (G, k) of CPOVD; else if it returns `No`, then we immediately conclude that no such solution that contains the vertex set P_i. Now at the end, if for each $P_i \in \mathcal{P}$, we have that no such solution exists that contains the vertex set P_i we return `No` for the input instance (G, k) of CPOVD. We provide the correctness of the above-described algorithm for an input instance (G, k) of CPOVD in the Section A.2.

Proof of Theorem 12. We show that our algorithm for CPOVD runs in $\mathcal{O}^*(14^k)$ time. Recall that the branching step is at most a 7-way branching, producing $\mathcal{O}^*(7^k)$ nodes in the branching tree. At each node of the branching tree, we spend only polynomial time, since detecting a K_3, C_4, or T_2 subgraph, or reporting their absence, can be done in polynomial time. Therefore, the total time spent in branching is $\mathcal{O}^*(7^k)$. Since there are $\mathcal{O}^*(7^k)$ leaves in the branching tree, the number of distinct root-to-leaf paths is also bounded by $\mathcal{O}^*(7^k)$, and thus $|\mathcal{P}| = \mathcal{O}^*(7^k)$. Consequently, we make at most $\mathcal{O}^*(7^k)$ calls to Group Steiner Tree. For any $P_i \in \mathcal{P}$, we call Group Steiner Tree only if $|P_i| + p \leq k$. By Theorem 13, each such Group Steiner Tree instance can be solved in $\mathcal{O}^*(2^{|P_i|+p})$, that is, in $\mathcal{O}^*(2^k)$ time. Therefore, solving all Group Steiner Tree instances requires at most $\mathcal{O}^*(7^k \cdot 2^k) = \mathcal{O}^*(14^k)$ time. Combining the branching and the Group Steiner Tree solving phases, the total running time is bounded by $\mathcal{O}^*(7^k) + \mathcal{O}^*(14^k) = \mathcal{O}^*(14^k)$. Hence, the proof. ◀

7 FPT Algorithm for IPOVD

In this section, we present an FPT algorithm for IPOVD with a running time of $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. This algorithm also begins by branching on occurrences of T_2, K_3, and C_4 subgraphs. Once this branching process finishes, we have at most 7^k root-to-leaf paths in the branching tree, and every minimal independent povd solution must fully include one of these paths. For each such path P_i, we check whether it can be extended to a solution to IPOVD, where a povd S is said to be a solution to IPOVD if $|S| \leq k$ and S is an independent set in G. This verification for each fixed P_i can be done in polynomial time.

Algorithm. We now describe the algorithm in more detail. Given an instance (G, k) of IPOVD, we iteratively remove occurrences of T_2, K_3, and C_4 by branching. Whenever such a structure exists in G, we branch over its vertices, reducing k by one at each branching step. Branching stops when we reach to an instance (G', k') where either $k' \leq 0$ or G' no longer

contains any of these subgraphs. Once the branching tree is constructed, we process each root-to-leaf path P_i as follows: we check whether P_i forms an independent set, and whether every cycle in $G - P_i$ has at least one vertex with no neighbors in P_i. If this is true, we select exactly one such vertex from each cycle and collect them into a set S_i. If $|S_i \cup V(P_i)| \leq k$, we return $S_i \cup V(P_i)$ as a valid IPOVD solution. If no such path extends to such a solution, we return that (G, k) is a `No` instance of IPOVD.

Correctness. As established earlier, there are at most 7^k different root-to-leaf paths in the branching tree. Let $\mathcal{P}$ be the collection of these paths. Every minimal IPOVD solution must completely contain one of the paths in $\mathcal{P}$. Therefore, for each $P_i \in \mathcal{P}$, we test whether it can be extended to a solution of size at most k. Observe that this requires P_i to be an independent set. Since $G - P_i$ contains no T_2, K_3, or C_4, by Fact 2 each connected component of $G - P_i$ is either a tree or a cycle (possibly with pendant vertices). Let c_i be the number of cycles in $G - P_i$. Any solution must pick at least one vertex from each such cycle. Therefore, constructing S_i by selecting one vertex per cycle with no neighbors in P_i suffices. Since the cycles are disjoint, we do not need to worry about independence among vertices in S_i. This establishes the correctness of the algorithm.

Running Time. Constructing the branching tree takes time $\mathcal{O}^*(7^k)$. For each of the 7^k paths P_i, checking whether it extends to a valid solution requires only polynomial time. Thus, the total time is $\mathcal{O}^*(7^k)$. Additionally, since a polynomial kernel of size $\mathcal{O}(k^3)$ exists for IPOVD, we first run kernelization on the input instance (G, k). If kernelization returns a trivial `Yes` or `No` instance, we return the corresponding answer `Yes` or `No`, respectively, to IPOVD. Otherwise, it produces an equivalent instance (G', k') with $\mathcal{O}(k^3)$ vertices, to which we apply the algorithm above. This results in a total running time of $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$. Hence, we obtain the following theorem.

▶ **Theorem 14.** IPOVD *admits an* FPT *algorithm with running time* $7^k \cdot k^{\mathcal{O}(1)} + n^{\mathcal{O}(1)}$.

8 Conclusion

We studied two variants of the POVD problem: the connected version (CPOVD) and the independent version (IPOVD). We showed that CPOVD is solvable in $\mathcal{O}^*(14^k)$ time but admits no polynomial kernel unless NP $\subseteq$ co-NP/poly. In contrast, IPOVD can be solved in $\mathcal{O}^*(7^k)$ time with a polynomial kernel of size $\mathcal{O}(k^3)$. Improving these running times, reducing the kernel size for IPOVD, improving the polynomial-time approximation factor, or establishing lower bounds are interesting avenues for future research.

References

1 Ankit Abhinav, Satyabrata Jana, Nidhi Purohit, Abhishek Sahu, and Saket Saurabh. Parameterized complexity of feedback vertex set with connectivity constraints. In Rastislav Královic and Vera Kurková, editors, *SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part I*, volume 15538 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2025. doi:10.1007/978-3-031-82670-2_3.

2 Akanksha Agrawal, Sushmita Gupta, Saket Saurabh, and Roohani Sharma. Improved algorithms and combinatorial bounds for independent feedback vertex set. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 2:1–2:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.2.

3 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discret. Math.*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

4 Fabián A. Chudak, Michel X. Goemans, Dorit S. Hochbaum, and David P. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Oper. Res. Lett.*, 22(4-5):111–118, 1998. doi:10.1016/S0167-6377(98)00021-2.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 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 Rafail Ostrovsky, editor, *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. doi:10.1109/FOCS.2011.23.

7 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. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. An improved FPT algorithm and a quadratic kernel for pathwidth one vertex deletion. *Algorithmica*, 64(1):170–188, 2012. doi:10.1007/s00453-011-9578-2.

9 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, 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 378–389. Springer, 2009. doi:10.1007/978-3-642-02927-1_32.

10 Baris Can Esmer and Ariel Kulik. Sampling with a black box: Faster parameterized approximation algorithms for vertex deletion problems. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.39.

11 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008. doi:10.1007/S00453-007-9145-Z.

12 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. doi:10.1007/s00453-021-00815-w.

13 Satyabrata Jana, Daniel Lokshtanov, Soumen Mandal, Ashutosh Rai, and Saket Saurabh. Parameterized approximation scheme for feedback vertex set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 56:1–56:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.56.

14 R. Krithika, Diptapriyo Majumdar, and Venkatesh Raman. Revisiting connected vertex cover: FPT algorithms and lossy kernels. *Theory Comput. Syst.*, 62(8):1690–1714, 2018. doi:10.1007/S00224-017-9837-Y.

15 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

16 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^\star(2.7k)$ time. *ACM Trans. Algorithms*, 18(4):34:1–34:26, 2022. doi:10.1145/3504027.

17 Shaohua Li and Marcin Pilipczuk. An improved FPT algorithm for independent feedback vertex set. *Theory Comput. Syst.*, 64(8):1317–1330, 2020. doi:10.1007/s00224-020-09973-w.

18 Ching-Hao Liu, Sheung-Hung Poon, and Jin-Yong Lin. Independent dominating set problem revisited. *Theor. Comput. Sci.*, 562:1–22, 2015. doi:10.1016/J.TCS.2014.09.001.

19 Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Covering small independent sets and separators with applications to parameterized algorithms. *ACM Trans. Algorithms*, 16(3):32:1–32:31, 2020. doi:10.1145/3379698.

20 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, 2013. doi:10.1145/2500119.

21 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theor. Comput. Sci.*, 461:65–75, 2012. doi:10.1016/j.tcs.2012.02.012.

22 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. doi:10.1007/s10878-011-9394-2.

23 Daniel Mölle, Stefan Richter, and Peter Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory Comput. Syst.*, 43(2):234–253, 2008. doi:10.1007/S00224-007-9089-3.

24 Geevarghese Philip, Venkatesh Raman, and Yngve Villanger. A quartic kernel for pathwidth-one vertex deletion. In Dimitrios M. Thilikos, editor, *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 196–207, 2010. doi:10.1007/978-3-642-16926-7_19.

25 Johannes Rauch, Dieter Rautenbach, and Uéverton S. Souza. Exact and parameterized algorithms for the independent cutset problem. *J. Comput. Syst. Sci.*, 148:103598, 2025. doi:10.1016/J.JCSS.2024.103598.

26 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

27 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

28 Yinglei Song. An improved parameterized algorithm for the independent feedback vertex set problem. *Theor. Comput. Sci.*, 535:25–30, 2014. doi:10.1016/j.tcs.2014.03.031.

29 Dekel Tsur. Faster algorithm for pathwidth one vertex deletion. *Theor. Comput. Sci.*, 921:63–74, 2022. doi:10.1016/j.tcs.2022.04.001.

A Appendix

A.1 Correctness of Reduction Rules 4-8

▶ **Lemma 15.** *Reduction Rule 4 is correct.*

Proof. Let P_v denote the set of all pendant neighbors of v in G, and let $D_v \subseteq P_v$ be the set of pendant neighbors of v that are deleted from G to obtain G'. Since G' is a subgraph of G and $R' \subseteq R$, if (G, R, k) is a `Yes` instance, then (G', R', k) is also a `Yes` instance. For the reverse direction, assume that (G', R', k) is a `Yes` instance. Then there exists a solution S for G' such that $|S| \leq k$ and $S \cap R' = \emptyset$. We claim that S is also a solution for G and satisfies $S \cap R = \emptyset$.

Suppose, for the sake of contradiction, that S is not a solution for G. Then there exists a structure T_2 (say, Q) in G such that $S \cap Q = \emptyset$. Since S is a solution for G', it follows that Q must contain some vertex $u \in Q \cap D_v$. Because $|S| \leq k$ and $|P_v \setminus D_v| = k + 1$, there exists a vertex $u' \in P_v \setminus D_v$ such that $u' \notin S$. Let $Q' = (Q \setminus \{u\}) \cup \{u'\}$. Then Q' is also a T_2 in G' satisfying $Q' \cap S = \emptyset$, which contradicts the assumption that S is a solution for G'.

Finally, since $S \cap D_v = \emptyset$, $R = R' \cup D_v$, and $S \cap R' = \emptyset$, we have $S \cap R = \emptyset$. Therefore, S is a solution for G of size at most k and disjoint from R, implying that (G, R, k) is a `Yes` instance. This completes the proof. ◀

▶ **Lemma 16.** *Reduction Rule 5 is correct.*

Proof. Notice that, in this case, if there is a solution of size at most k, then that solution must include u. Otherwise, to eliminate all the T_2 in $G[V(M) \cup \{u\}]$, we would need to delete strictly more than k vertices. Thus, if $u \in R$, there cannot be a k size solution disjoint from R. On the other hand, if $u \notin R$, then deleting u, decreasing k by one, and adding $N(u)$ to R is safe, since u necessarily belongs to any valid solution. ◀

▶ **Lemma 17.** *Reduction Rule 6 is correct.*

Proof. First, we assume that (G, R, k) is a `Yes` instance. We want to show that (G', R', k) is also a `Yes` instance. Let S be an ipovd of size at most k disjoint from R. There are two cases to consider: either $S \cap \{v_3, v_4\} = \emptyset$ or exactly one of v_3 and v_4 belongs to S. If $S \cap \{v_3, v_4\} = \emptyset$, then the same set S is an ipovd for G' of size at most k disjoint from R, since the contraction does not create any new cycles or new T_2. Next, suppose exactly one of v_3 or v_4 belongs to S. Then the contracted vertex $v_3^4 \notin R'$ in G'. Notice that v_3 and v_4 cannot be part of any T_2, and if they belong to any cycle, then v_2, v_3, v_4, and v_5 all participate in that cycle. Therefore, if either v_2 or v_5 is in S, then the set $S \setminus \{v_3, v_4\}$ is an ipovd for G' of size at most k disjoint from R. Otherwise, the set $S' = \big(S \setminus \{v_3, v_4\}\big) \cup \{v_3^4\}$ is an ipovd for G' of size at most k disjoint from R'.

In converse direction, assume that (G', R', k) is a `Yes` instance, and let S' be a solution for (G', R', k) of size at most k. If $v_3^4 \notin S'$, then by Observation 7, the set $S = S'$ is a solution for (G, R, k) of size at most k. Otherwise, if $v_3^4 \in S'$, then at least one of v_3 or v_4 was not in R in G. Let that vertex be v_i, where i is either 3 or 4. Then again by Observation 7, the set $S = \big(S' \setminus \{v_3^4\}\big) \cup \{v_i\}$ is a solution for (G, R, k) of size at most k. ◀

▶ **Lemma 18.** *Reduction Rule 7 is correct.*

Proof. The correctness of Reduction Rule 7 is quite similar to the correctness of Reduction Rule 5, since any POVD solution must cover all cycles, and thus any k size POVD solution must include u in this case. Therefore, if $u \in R$, there cannot be a k size POVD solution. Otherwise, if $u \notin R$, then deleting u, decreasing k by one, and adding $N(u)$ to R is safe, as u necessarily belongs to any valid solution. ◀

▶ **Lemma 19.** *Reduction Rule 8 is correct.*

Proof. First, we will show that if S is a minimal ipovd of size k for G disjoint from R, then S is also a ipovd of size k for G' and remains disjoint from R'. To establish this, it suffices to show that $S \cap Y' = \emptyset$.

For contradiction, suppose that $y_i^x \in S$ for some $y_i^x \in Y'$. Since S is an independent set, both u and x are not in S. Also observe that in $G\big[V \setminus \big(S \setminus \phi(x)\big)\big]$, there are $k+2$ vertex-disjoint paths between u and x, each passing through a different vertex of $\phi(x)$. This implies that at least $k+1$ vertices of $\phi(x)$ must be included in S, which contradicts the assumption that $|S| \leq k$. Therefore, we have $S \cap Y' = \emptyset$.

For the converse direction, let S' be an ipovd of size k for the graph G', disjoint from R'. We need to show that S' is an ipovd for G as well. Note that by the way Reduction Rule 8 has been applied, for each $x \in X'$, there exist indices i and j such that the cycle (u, y_i^x, x, y_j^x, u) is present in G'. To hit this cycle, at least one of the vertices u or x must belong to S'. Since $S' \cap Y' = \emptyset$, the set S' is independent and remains disjoint from R.

Now, suppose for contradiction that S' is not a ipovd in G. Then there must be some cycle or T_2 in $G[V \setminus S']$. Since there was no such cycle or T_2 in $G'[V \setminus S']$, any such structure in G must include an edge of the form uy_i^x for some i. However, for each x, there are exactly two such edges in G', and their presence would imply a cycle or T_2 in G', contradicting our assumption. Hence, S' is indeed a ipovd of size k for G disjoint from R. ◀

A.2 Missing proof details for Theorem 12

Correctness. We will argue the correctness of the above-described algorithm for an input instance (G, k) of CPOVD in the following two claims.

▷ Claim 20. The algorithm returns `Yes` if and only if (G, k) is a `Yes` instance.

Proof. If the algorithm returns `Yes`, then at least one call to Group Steiner Tree must have returned `Yes`, corresponding to some $P_j \in \mathcal{P}$. If S is a solution to the Group Steiner Tree instance for this call, then by the way the instance was constructed, we have $P_j \subseteq S$, and S also contains at least one vertex from each cycle in $G - P_j$. Thus, $G - S$ is acyclic and T_2-free (since $G - P_j$ is K_3, C_4, and T_2-free). Moreover, by the properties of the Group Steiner Tree solution corresponding to P_j, $G[S]$ is connected and $|S| \leq k$. Hence, S is a valid solution for CPOVD.

If the algorithm returns `No` because for each $P_i \in \mathcal{P}$ either $|P_i| + p_i > k$ or the algorithm return `No` when it call to corresponding Group Steiner Tree. In the first case, if $|P_i| + p_i > k$, then there cannot be a size-k pathwidth-one vertex deletion set, since any such set must include all vertices of some P_i together with at least one vertex from each of the p cycles in $G - P_i$, exceeding the budget k. If second case, the algorithm must made a call to Group Steiner Tree and returned `No`, Towards contradiction, suppose there exists a solution S to CPOVD for (G, k). Then there must be some $P_j \in \mathcal{P}$ such that $P_j \subseteq S$. Moreover, S must contain at least one vertex from each cycle in $G - P_j$. Since S is a solution to CPOVD, we have that $G[S]$ is connected and $|S| \leq k$. This implies that S would be a solution to the Group Steiner Tree instance corresponding to P_j, contradicting the assumption that every call to Group Steiner Tree returned `No`. ◁

Improved Upper Bounds on Multiflow-Multicut Gaps in Cactus Graphs

Sina Kalantarzadeh
University of Waterloo, Waterloo, Canada

Nikhil Kumar
Tata Institute of Fundamental Research, Mumbai, India

Abstract

Given a set of source-sink pairs, the *maximum multiflow* problem asks for the largest total amount of flow that can be feasibly routed between them. The *minimum multicut* problem, which is dual to multiflow, seeks the lowest-cost set of edges whose removal disconnects all source-sink pairs. It is straightforward to see that the value of a minimum multicut is at least that of the corresponding maximum multiflow. The ratio between the two is known as the *multiflow-multicut gap*. The classical max-flow min-cut theorem tells us that this gap is exactly one when there is only a single source-sink pair. However, for multiple source-sink pairs, the gap can be arbitrarily large. In this work, we investigate the multiflow-multicut gap in *cactus graphs*, and establish the following results (i) *tight* upper bound of 1.5 for *cycle* (ii) an upper bound of $2 + \frac{2}{\ln 2} < 3.45$ for general *cactus* graph (iii) *tight* upper bound of 2 for *unicyclic graphs*, where the graph contains exactly one cycle (iv) *tight* upper bound of 2 for *path cactus graphs*, where cycles are arranged along a single path. We develop novel generalizations of the classical rounding algorithm to establish our results.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Approximation algorithms

Keywords and phrases Approximation Algorithms, Randomized Algorithms, Linear Programming, Graph Algorithms, Multicut, Multicommodity flow

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.40

Funding The work was done while the second author was a postdoctoral researcher at the University of Waterloo and was supported in part by J. Cheriyan's NSERC Discovery Grant RGPIN-2024-04473 and C. Swamy's NSERC Discovery Grant RGPIN-2024-04532.

Acknowledgements We thank Joseph Cheriyan for many helpful discussions throughout this project. We are also grateful to Hadas Barabash and Tom Iagovet for their work and insights during Summer 2024, which contributed to these results.

1 Introduction

Given an edge-weighted graph with k source-sink pairs, a *multicut* is a set of edges whose removal separates each source from its corresponding sink. The *minimum multicut problem* aims to find such a multicut with the smallest possible total edge weight. This problem generalizes the classical minimum s-t cut problem and has been extensively studied. Notably, computing the minimum multicut is NP-hard, even on simple graph classes such as trees [10].

Closely related to the multicut problem is the *multicommodity flow* problem (or *multiflow*), where the objective is to maximize the total flow that can be simultaneously routed between the source-sink pairs. When the flow is required to be integral, the problem becomes the *maximum integer multiflow problem*, a generalization of the well-known *edge-disjoint paths* problem. Since every source-sink path must cross some edge in any feasible multicut, the value of any multicut provides an upper bound on the value of any feasible multiflow.


© Sina Kalantarzadeh and Nikhil Kumar;
licensed under Creative Commons License CC-BY 4.0

In fact, the natural linear programming (LP) relaxation of the multicut problem is the dual of the LP formulation of the multiflow problem. The ratio between the cost of the minimum multicut and the value of the maximum multicommodity flow is known as the *multiflow–multicut gap*. By LP strong duality, this ratio also bounds the integrality gap of the LP relaxation of the multicut problem, and vice versa.

The celebrated max-flow min-cut theorem [7] states that the multiflow–multicut gap is exactly 1 when $k = 1$, i.e., for a single source-sink pair. A classic result by Hu [11] extends this and shows that the gap remains 1 when $k = 2$. However, the equality fails when $k \geq 3$, even on very simple graphs (see [10] for an example).

Garg, Vazirani, and Yannakakis [9] established a tight bound of $\Theta(\log k)$ on the multiflow–multicut gap in general graphs. For trees, the gap is exactly 2 [10]. For K_r-minor-free graphs, Tardos and Vazirani [16] used the decomposition theorem of Klein, Plotkin, and Rao [15] to obtain an upper bound of $\mathcal{O}(r^3)$. Building on a long sequence of results [5, 1, 8, 6], Conroy and Filtser [3] recently proved an asymptotically tight bound of $\Theta(\log r)$ on the multiflow–multicut gap for K_r-minor-free graphs.

The primary motivation behind the aforementioned works was to establish asymptotic bounds on the integrality gap (as a function of r), without focusing on optimizing the constants involved. However, for specific graph families, such as planar graphs – the constants obtained from these results are quite large (close to 100). As a result, determining the exact integrality gap remains an intriguing open question. Tighter upper and lower bounds are still elusive and serve as the central motivation for this paper.

While the multiflow–multicut gap is well understood for trees, in this work we study a broader class of graphs, namely *cactus* graphs. Recall that in a cactus graph, any two simple cycles are either vertex-disjoint or share exactly one vertex. Recently, [14] showed that the multiflow–multicut gap for cactus graphs is at least $\frac{16}{7}$. We complement their result by proving improved upper bounds. In particular, we show that the multiflow–multicut gap is at most 3.45 for cactus graphs, improving upon the bound of 4 by Bentz [2]. Our approach involves a novel modification of the well-known algorithm for trees. Moreover, we demonstrate that adding structural constraints to cactus graphs can yield even tighter bounds. Specifically, we prove a *tight* upper bound of 2 for *unicyclic graphs*, graphs with exactly one cycle, which we refer to as the *central cycle*, and also a *tight* upper bound of 2 for *path cactus graphs*, where cycles are arranged along a single path. For completeness, we also include a tight integrality gap of $\frac{3}{2}$ for simple cycles – likely known in the community, although we could not find an explicit reference.

All our results are derived using the fact that the natural linear programming relaxations of the multicut and multiflow problems are dual to each other. We introduce new algorithmic ideas to round the linear programming relaxation to the multicut problem, and then use strong duality to obtain our bounds.

2 Preliminaries

Given a connected graph G, we denote its vertex and edge sets by $V(G)$ and $E(G)$, respectively. We will use K_r to denote the complete graph on r vertices. In this paper, we will only be concerned with cactus graphs. Graphs in which every edge is contained in at most one cycle are called *cactus* graphs. Cactus graphs are a subclass of series-parallel and planar graphs, and are arguably the simplest family of planar graphs after trees and cycles. Cactus and series-parallel graphs do not contain K_4 as a minor.

Let G be a simple undirected graph with edge costs $c\colon E(G) \to \mathbb{R}_{\geq 0}$, and let $\{(s_i, t_i)\}_{i=1}^{k}$ be the set of source-sink pairs. Let $\mathcal{P}_i$ denote the set of all paths between s_i and t_i in G, and let $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_i$. A *multicut* is a set of edges $F \subseteq E(G)$ such that every $P \in \mathcal{P}$ contains at least one edge in F. Equivalently, a multicut is a set of edges whose removal disconnects every source-sink pair.

Given two arbitrary vertices $u, v \in V(G)$, we use $d_G(u, v)$ to denote the shortest path distance between u and v in G. The diameter of G is the maximum distance between a pair of vertices in G, i.e., $\mathrm{diam}(G) = \max_{u,v \in V(G)} d_G(u, v)$. We use $d_G(v, e)$ to denote the distance of a vertex v from an edge $e = (x, y)$, i.e., $d_G(v, e) = \min\{d_G(v, x), d_G(v, y)\}$.

For $F \subseteq E(G)$, we use $G \setminus F$ to denote the graph obtained after the removal of F from G. For any $v \in V(G)$, we use $C_F(v)$ to denote the connected component of $G \setminus F$ containing v. We overload notation and also use $C_F(v)$ to denote the set of vertices in the connected component containing v. We define the radius of v with respect to F as the distance of the farthest vertex from v in $C_F(v)$, i.e., $\mathrm{rad}_F(v) = \max_{u \in C_F(v)} d_G(v, u)$. In addition, the diameter of F is the maximum diameter of a connected component after the removal of F from G, i.e., $\mathrm{diam}(F) = \max_{v \in V(G)} \mathrm{diam}(C_F(v))$. Given $t \in \mathbb{R}_{\geq 0}$ as a parameter, we say that F forms a t-diameter decomposition if $\mathrm{diam}(F) < t$. We denote the set of all t-diameter decompositions of G by $\mathcal{F}_t(G)$. Note that when referring to the distance between two vertices u, v in a component C, $d_G(u, v)$ denotes their distance in G, rather than in the subgraph induced by C, i.e., $G[C]$. Given a cycle C and a path P, we denote by $\mathrm{len}(C)$ and $\mathrm{len}(P)$ the number of edges in C and P, respectively.

▶ **Definition 1.** *Let $P = (v_1, \ldots, v_n)$ be a path on n vertices. For any $i \neq j$, we denote by $P[v_i, v_j]$ the induced subpath of P between v_i and v_j (inclusive).*

▶ **Definition 2.** *Let $P = (v_1, \ldots, v_n)$ and $Q = (v_n, \ldots, v_{n+m})$ be two distinct paths such that they share the vertex v_n. We denote by $P \oplus Q$ the path from v_1 to v_{n+m} obtained by concatenating P and Q.*

2.1 Graphs Classes

We recall a few definitions from the graph theory textbook by Diestel [4]. A *block* is a maximal connected subgraph without a cut-vertex (of that subgraph). Thus, every block is either a maximal 2-connected subgraph, or a cut-edge (with its ends), or an isolated vertex. The *block graph* of a graph G is a bipartite graph on $A \cup B$, where A is the set of cut-vertices of G and B is the set of blocks of G, and the block graph has an edge ab (where $a \in A$, $b \in B$) iff $a \in V(B)$.

A *cactus* is a connected graph in which every block is either a cycle or a cut-edge (with its ends). A *path-cactus* is a cactus whose block graph is a path and that has no cut-edges. A *unicyclic* graph is a connected graph with exactly one cycle. We denote these families by CACTUS, PATH CACTUS, UNICYCLIC, and CYCLE (the last being the family of all simple cycles).

2.2 Linear Programming Relaxation for the Minimum Multicut Problem

We begin by describing an integer programming (IP) formulation for the minimum multicut problem. For each edge $e \in E(G)$, we introduce an integer variable $x(e) \in \{0, 1\}$, which indicates whether the edge is selected in the multicut. For a given path P, we define $x(P) = \sum_{e \in E(P)} x(e)$. A feasible multicut must include at least one edge from each source-sink path, so we impose the constraint $x(P) \geq 1$ for all $P \in \mathcal{P}$, ensuring that each path is *cut* by at least one edge. We relax the integrality constraints to obtain the linear programming (LP) relaxation (1) of the multicut problem, which is formulated as follows:

$$\begin{aligned} &\min \sum_{e \in E(G)} c(e) \cdot x(e) \\ &\textit{subject to} \\ &\quad x(P) \geq 1 \quad \forall P \in \mathcal{P} \\ &\quad x(e) \geq 0 \quad \forall e \in E(G). \end{aligned} \tag{1}$$

Even though there are an exponential number of constraints, it is well known that the optimal solution to this LP can be computed in polynomial time [9]. We denote the optimal solutions of the integer and linear programs as OPT_{IP} and OPT_{LP}, respectively. We refer to OPT_{LP} as the minimum fractional multicut. We know that the value of the maximum multiflow is equal to the minimum fractional multicut. Furthermore, a bound on the integrality gap of the LP relaxation for the multicut problem provides the same bound for the multiflow-multicut gap. Therefore, from this point onward, we will focus solely on the integrality gap of the multicut LP.

▶ **Definition 3.** *Let $\mathcal{G}$ be a family of graphs, and let $\mathcal{M}(\mathcal{G})$ be the family of all instances of the minimum multicut problem on $\mathcal{G}$, obtained by assigning arbitrary capacities to the edges and selecting a set of source-sink pairs. The integrality gap $\alpha_{\mathcal{M}(\mathcal{G})}$ of the minimum multicut problem on $\mathcal{M}(\mathcal{G})$ is defined as follows:*

$$\alpha_{\mathcal{M}(\mathcal{G})} := \max_{M \in \mathcal{M}(\mathcal{G})} \frac{OPT_{IP}(M)}{OPT_{LP}(M)}.$$

From the discussion above, we know that $\alpha_{\mathcal{M}(\text{TREE})} = 2$ [10], where TREE denotes the family of all trees, and $\alpha_{\mathcal{M}(\text{PLANAR})} = \mathcal{O}(1)$ [15], where PLANAR refers to the family of all planar graphs. In this paper, we focus on cactus graphs.

2.3 Transition from the Minimum Multicut Problem to the Small Diameter Decomposition Problem

For all of our upper bounds, we reduce the analysis of the multicut LP to a Small Diameter Decomposition (SDD) guarantee. We first define SDD formally.

▶ **Definition 4** (Small Diameter Decomposition (SDD))**.** *Given an unweighted graph G, an integer $k \in \mathbb{N}$, and a parameter $p \in (0, 1)$, an SDD for (G, k, p) is a probability distribution $\mathcal{D} = \{y_F\}_{F \in \mathcal{F}_k(G)}$ over the family $\mathcal{F}_k(G)$ of k-diameter decompositions of G such that every edge $e \in E(G)$ is included in a random k-diameter decomposition sampled from $\mathcal{D}$ with probability at most p:*

$$\sum_{\substack{F \in \mathcal{F}_k(G) \\ e \in F}} y_F \;\leq\; p \qquad \textit{for all } e \in E(G).$$

If such a distribution exists, we write $SDD(G, k, p)$.

We use the connection between SDDs and the multicut integrality gap established in [14]. We only need the "backward" direction for upper bounds on the integrality gap (the "forward" direction was used in [14] to derive the lower bound 16/7 for the family CACTUS).

▶ **Theorem 5** (Theorem 2 of [14])**.** *Let $\mathcal{G}$ be a family of graphs closed under taking minors and under edge subdivisions, and let $\alpha > 0$. Then*

$$\alpha_{\mathcal{M}(\mathcal{G})} \leq \alpha \quad \Longleftrightarrow \quad \forall\, G \in \mathcal{G},\ \forall\, w \in \mathbb{N},\ \exists\, SDD\big(G, 2w, \tfrac{\alpha}{2w}\big)\,.$$

Transitioning to the SDD framework removes any dependence on the placement of source–sink pairs and edge costs, and replaces it with a uniform, metric-style requirement.
All graph families we consider (cycles, unicyclic graphs, path-cactus graphs, and cactus graphs) are closed under minors and edge subdivisions. It is convenient to work with instances in which every cycle of the graph has even length. More generally, for any $k \in \mathbb{N}$, Lemma 6 shows that establishing an SDD with parameters $(4kw, \alpha/(4kw))$ for the normalized instances (where each cycle length is divisible by $2k$) suffices to obtain an SDD with parameters $(2w, \alpha/(2w))$ for the original family.

▶ **Lemma 6** (see Appendix B for the proof in the appendix). *Fix $k \in \mathbb{N}$ and $\alpha > 0$. Let $\mathcal{G}$ be a subfamily of cactus graphs closed under minors and edge subdivisions, and let*

$$\mathcal{G}' \;=\; \{\, G' \in \mathcal{G} \;:\; \textit{every cycle of } G' \textit{ has length divisible by } 2k \,\}.$$

Then

$$\Big[\forall\, G' \in \mathcal{G}',\, \forall\, w \in \mathbb{N},\, \exists\, SDD\big(G', 4kw, \tfrac{\alpha}{4kw}\big)\Big] \implies \Big[\forall\, G \in \mathcal{G},\, \forall\, w \in \mathbb{N},\, \exists\, SDD\big(G, 2w, \tfrac{\alpha}{2w}\big)\Big].$$

In what follows, we apply Theorem 5 together with Lemma 6 to obtain the stated upper bounds for the families CYCLE, UNICYCLIC, PATH CACTUS, and CACTUS. By Lemma 6, we may assume that every cycle in G has even length (the case $k = 1$ of the normalization). For such graphs, it suffices to show that for any $w \in \mathbb{N}$ there exists an $SDD(G, 2w, \frac{\alpha}{2w})$.

Throughout, the setting is as follows: given a graph G from the relevant subfamily with all cycles even (by Lemma 6) and a parameter $w \in \mathbb{N}$, we construct an $SDD(G, 2w, \frac{\alpha}{2w})$. This certifies the required SDD and, by Theorem 5, yields the upper bound α on the integrality gap. To prove existence, we design randomized algorithms that output a $2w$-diameter decomposition while controlling the marginal probability that any edge is cut. We fix w and present four algorithms, each tailored to one graph class. Given G, an algorithm samples a set $F \subseteq E(G)$ such that every component of $G \setminus F$ has diameter at most $2w - 1$ (i.e., $F \in \mathcal{F}_{2w}(G)$) and we bound $\Pr[e \in F]$ for each edge. We will use Theorem 5 and Lemma 6 repeatedly in what follows.

- **Algorithm 1:** For a cycle $C \in$ CYCLE of even length, each edge $e \in E(C)$ is removed with probability at most $\frac{3}{4} \cdot \frac{1}{w}$. By Theorem 5, this yields $\alpha_{\text{CYCLE}} \le 2 \cdot \frac{3}{4} = \frac{3}{2}$. A matching lower bound (see Appendix A in the appendix) shows that $\alpha_{\text{CYCLE}} = \frac{3}{2}$.
- **Algorithm 2:** For a cactus graph $G \in$ CACTUS in which every cycle has even length, each edge $e \in E(G)$ is removed with probability at most $\big(1 + \frac{1}{2\ln 2}\big) \cdot \frac{1}{w} \approx 1.721 \cdot \frac{1}{w}$. Hence there exists an $SDD(G, 2w, 1.721 \cdot \frac{1}{w})$, and Theorem 5 together with Lemma 6 implies

  $$\alpha_{\text{CACTUS}} \le 2\big(1 + \tfrac{1}{2\ln 2}\big) = 2 + \tfrac{1}{\ln 2} \approx 3.443 \;<\; 3.45.$$

- **Algorithm 3:** For a unicyclic graph $G \in$ UNICYCLIC, each edge $e \in E(G)$ is removed with probability at most $\frac{2}{2w-1}$. If this were $\frac{2}{2w}$ we would immediately get $\alpha_{\text{UNICYCLIC}} \le 2$; since it is slightly larger, we apply Lemma 6 to obtain $\alpha_{\text{UNICYCLIC}} \le 2$ (see Corollary 25). Because unicyclic graphs contain trees as subgraphs (for which the gap is 2 [10]), we conclude $\alpha_{\text{UNICYCLIC}} = 2$.
- **Algorithm 4:** For a path-cactus graph $G \in$ PATH-CACTUS in which all cycles have even length, each edge $e \in E(G)$ is removed with probability at most $\frac{1}{w}$. Hence there exists an $SDD(G, 2w, \frac{1}{w})$, and Theorem 5 and Lemma 6 yield $\alpha_{\text{PATH-CACTUS}} \le 2$.
  In the appendix (see Appendix C in the appendix), we give an explicit instance with gap at least 2, and therefore $\alpha_{\text{PATH-CACTUS}} = 2$.

2.4 Small Diameter Decomposition for Trees

As mentioned earlier, $\alpha_{\mathcal{M}(\text{TREE})} = 2$. By Theorem 5, this implies that for any tree T and any $w \in \mathbb{N}$,

$$SDD\big(T, 2w, \tfrac{2}{2w}\big) = SDD\big(T, 2w, \tfrac{1}{w}\big)\,.$$

Here we give an explicit construction of $SDD(T, 2w, \frac{1}{w})$ without invoking Theorem 5. This construction will be reused in Section 5 to obtain an appropriate SDD for unicyclic graphs.

▶ **Theorem 7.** *Let T be a tree. Then for every integer $w \in \mathbb{N}$, there exists $SDD\big(T, 2w, \frac{1}{w}\big)$. Equivalently, there exists a probability distribution $\mathcal{D} = \{y_F\}_{F \in \mathcal{F}_{2w}(T)}$ over $\mathcal{F}_{2w}(T)$ such that*

$$\sum_{\substack{F \in \mathcal{F}_{2w}(T) \\ e \in F}} y_F \;\le\; \frac{1}{w} \quad \forall e \in E(T). \tag{2}$$

Proof. Root the tree T at an arbitrary vertex $r \in V(T)$. For $i = 0, \ldots, w-1$, define

$$F_i = \{\, e \in E(T) \mid d(r, e) = i + kw \text{ for some } k \in \mathbb{Z}_{\ge 0} \,\}.$$

Set $y_{F_i} = \frac{1}{w}$ for each $i = 0, \ldots, w-1$, and $y_F = 0$ otherwise. Note that the sets F_i partition $E(T)$: we have $E(T) = \bigcup_{i=0}^{w-1} F_i$ and $F_i \cap F_j = \emptyset$ for $i \neq j$. Thus,

$$\sum_{\substack{F \in \mathcal{F}_{2w}(T) \\ e \in F}} y_F = \sum_{\substack{i=0 \\ e \in F_i}}^{w-1} y_{F_i} = \tfrac{1}{w}, \quad \forall e \in E(T).$$

It remains to show that each F_i is a valid $2w$-diameter decomposition. Fix F_i, and consider a pair of vertices (u, v) with $d(u, v) \ge 2w$. Let q be the lowest common ancestor of u and v. The unique u–v path consists of the u–q path and the q–v path. Since $d(u, v) \ge 2w$, one of these subpaths has length at least w. Without loss of generality, suppose $d(q, v) \ge w$, and denote this path by $Q = e_0, e_1, \ldots, e_p$. Because q is an ancestor of v, we have $d(r, e_i) = d(r, e_{i-1}) + 1$ for $i = 1, \ldots, p$. Hence there exists some $e_j \in Q$ such that $d(r, e_j) \equiv i \pmod{w}$, i.e., $e_j \in F_i$. Removing F_i therefore separates u and v, as required. This shows that F_i defines a $2w$-diameter decomposition, and hence $\mathcal{D}$ is a valid $SDD(T, 2w, 1/w)$. ◀

The $2w$-diameter decompositions $F_0, \ldots, F_{w-1}$ described in the proof of Theorem 7 will be useful in the **Algorithm 3** in Section 5, so we record a formal definition:

▶ **Definition 8.** *Let $w \in \mathbb{N}$, and let T be a tree with a distinguished root vertex $r \in V(T)$. For each $i = 0, 1, \ldots, w-1$, define*

$$F_w^i(T, r) := \{e \in E(T) \mid d_T(r, e) \equiv i \pmod{w}\}\,,$$

where $d_T(r, e)$ denotes the distance from r to the closer endpoint of e. Then $\{F_w^i(T, r)\}_{i=0}^{w-1}$ forms a partition of $E(T)$. Moreover, each $F_w^i(T, r)$ defines a $2w$-diameter decomposition of T, and the connected component containing the root r has radius at most i from r, that is,

$$rad_{F_w^i(T,r)}(r) \le i \le w - 1.$$

Algorithm 1 Cycle Algorithm.

1: Let $F = \varnothing$.
2: Choose an edge $e \in E(C)$ uniformly at random. Add e to F, and remove e from C.
3: Denote the path $C \setminus e$ as P, and let $r \in V(P)$ denote the starting-point of P.
4: Pick $\rho \in \{0, 1, \ldots, 2w-1\}$ uniformly at random.
5: Add $F^{\rho}_{2w}(P, r)$ to F, and remove $F^{\rho}_{2w}(P, r)$ from P.
6: Return F.

3 Cycle

Let $w \in \mathbb{N}$ and let C be a cycle of even length. If $len(C) \leq 4w-1$, then $\mathrm{diam}(C) \leq 2w-1$ and no edges need be removed (set $F = \emptyset$). Otherwise ($len(C) \geq 4w$), Algorithm 1 outputs a random $2w$-diameter decomposition $F \in \mathcal{F}_{2w}(C)$ such that, for every edge $e \in E(C)$, $\Pr[e \in F] \leq \frac{3}{4} \cdot \frac{1}{w}$.

▶ Remark 9. The choice of ρ in the line 4 is independent of the choice of the edges in the line 2 of Algorithm 1.

▶ **Lemma 10.** $F \in \mathcal{F}_{2w}(C)$.

Proof. Let P be the path referenced in line 3 of Algorithm 1, and let r denote its starting point. For all $i = 1, \ldots, 2w-1$, observe that $F^{i}_{2w}(P, r)$ is a $2w$-diameter decomposition of P. Therefore, F is a $2w$-diameter decomposition of C. ◀

Each edge $e \in E(C)$ is added to F either in the line 2, or the line 5. The first case occurs with probability $\frac{1}{len(C)} \leq \frac{1}{4w}$, and the second case occurs with probability $\frac{1}{2w}$. Thus, using the union bound, the probability of e being removed by Algorithm 1 is at most $\frac{1}{4w} + \frac{1}{2w} = \frac{3}{4} \cdot \frac{1}{w}$. Theorem 5 implies that $\alpha_{CYCLE} \leq 2 \times \frac{3}{4} = \frac{3}{2}$.

4 Cactus

Let $w \in \mathbb{N}$ and G be a cactus with distinct cycles $C_1, \ldots, C_k$ such that $len(C_i)$ is an even number for $i = 1, \ldots, k$. For simplicity, we denote the distance function d_G as d throughout this section. Let $r \in V(G)$ be an arbitrary vertex, which we denote as the root. For any cycle C_i, let $r_{C_i} \in V(C_i)$ be the unique vertex such that $d(r, r_{C_i}) = \min_{v \in V(C_i)} d(r, v)$, and let

$$\beta_i = 2 \cdot \sum_{j=1}^{len(C_i)/2} \frac{1}{len(C_i) - j},$$

and let $\mathcal{D}_i$ be the probability distribution on $E(C_i)$ defined by

$$\Pr_{\mathcal{D}_i}(e) = \frac{1}{\beta_i} \cdot \frac{1}{len(C_i) - d(r_{C_i}, e) - 1}, \quad \forall e \in E(C_i),$$

where β_i is the normalizing constant ensuring that the total probability sums to 1. To see this note that for each $j = 1, \ldots, \frac{len(C_i)}{2}$, we have $|\{e \in E(C_i) \mid d(r_{C_i}, e) = j-1\}| = 2$. The algorithm is described in detail in Algorithm 2.

▶ **Lemma 11.** $F \in \mathcal{F}_{2w}(G)$.

Algorithm 2 Cactus Algorithm.

1: Let $F = \varnothing$.
2: **for** $i = 1, \ldots, k$ **do**
3: Choose an edge $e_i \in E(C_i)$ randomly according to the distribution $\mathcal{D}_i$. Add e_i to F, and remove e_i from C_i.
4: **end for**
5: Let T denote the resulting graph, which is a tree.
6: Pick $\rho \in \{0, 1, \ldots, w-1\}$ uniformly at random.
7: Add $F_w^\rho(T, r)$ to F, and remove $F_w^\rho(T, r)$ from T.
8: **for** $i = 1, \ldots, k$ **do**
9: **if** the endpoints of e_i are within the same connected component in $G \setminus F$ **then**
10: Remove e_i from F, and keep e_i in G.
11: **end if**
12: **end for**
13: Return F.

Proof. Let F' denote the set of edges in F immediately before the execution of the line 8 of Algorithm 2. At that point, we have $F' = (E(G) \setminus E(T)) \cup F_w^\rho(T, r)$, which means that $F' \in \mathcal{F}_{2w}(G)$, as $F_w^\rho(T, r) \in \mathcal{F}_{2w}(T)$ mentioned in Definition 8. During the execution of the lines 8 to 12 of Algorithm 2, each edge e_i is removed from F' only if its endpoints are still connected in $G \setminus F'$. This means that no two previously disconnected components become reconnected. Thus, $F \in \mathcal{F}_{2w}(G)$. ◀

Now, we show that each edge $e \in E(G)$ is removed by Algorithm 2 with probability at most $(1 + \frac{1}{2 \ln 2}) \cdot \frac{1}{w}$.

▶ Remark 12. The choice of ρ in the line 6 is independent of the choice of the edges in the lines 2 to 4 of Algorithm 2.

Remark 12 implies that each cut-edge $e \in E(G)$ is removed by Algorithm 2 with probability $\frac{1}{w}$. For edges that lie within cycles, we have the following Lemma 13.

▶ **Lemma 13.** *Let $e \in E(C_i)$ for some $i = 1, \ldots, k$.* $\Pr[e \in F] = (1 + \frac{1}{\beta_i}) \cdot \frac{1}{w}$.

Proof. Let A be the event that $e \in F$, and let B be the event that e is chosen as e_i in line 3 of Algorithm 2. Then,

$$\Pr[A] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \mid \bar{B}] \cdot \Pr[\bar{B}].$$

By Remark 12, we have

$$\Pr[A \mid \bar{B}] \cdot \Pr[\bar{B}] \leq \frac{1}{w} \cdot 1 = \frac{1}{w}.$$

To compute $\Pr[A \mid B]$, let P_1 and P_2 denote the shorter and longer paths, respectively, between r_{C_i} and e in C_i (see Figure 1). Note that

$$\mathrm{len}(P_1) = d(r_{C_i}, e) < \mathrm{len}(P_2) = \mathrm{len}(C_i) - d(r_{C_i}, e) - 1.$$

Since r_{C_i} is the least common ancestor (LCA) of P_1 and P_2 in the tree T rooted at r (as defined in line 5), it follows that if $E(P_1) \cap F_w^\rho(T, r) \neq \varnothing$, then $E(P_2) \cap F_w^\rho(T, r) \neq \varnothing$ as well. Each edge in $E(P_2)$ is removed independently with probability $\frac{1}{w}$. Therefore, by the union bound, $\Pr[A \mid B] \leq \frac{\mathrm{len}(P_2)}{w}$.

Now, observe that

$$\Pr[A \mid B] \cdot \Pr[B] = \frac{\text{len}(C_i) - d(r_{C_i}, e) - 1}{w} \cdot \frac{1}{\beta_i} \cdot \frac{1}{\text{len}(C_i) - d(r_{C_i}, e) - 1} = \frac{1}{\beta_i w}.$$

Combining both parts, we conclude:

$$\Pr[A] = \frac{1}{\beta_i w} + \frac{1}{w} = \left(1 + \frac{1}{\beta_i}\right) \cdot \frac{1}{w}.$$

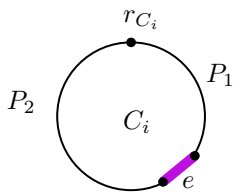


Figure 1 P_1, P_2, e in C_i.

◀

▶ **Lemma 14.** *$\beta_i \geq 2 \ln 2$ for all $i = 1, \ldots, k$.*

Proof. It is sufficient to prove $f(k) = \frac{1}{k} + \cdots + \frac{1}{2k-1} \geq \ln 2$ for any $k \in \mathbb{N}$. $f(k)$ is decreasing in $\mathbb{N}$. We have

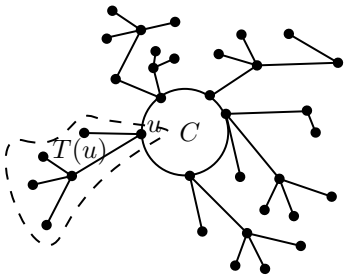
$$\sum_{i=k}^{2k-1} \frac{1}{i} \geq \int_k^{2k} \frac{1}{x}\, dx = \ln(2k) - \ln(k) = \ln\left(\frac{2k}{k}\right) = \ln 2.$$ ◀

Thus, the probability that any edge is removed by Algorithm 2 is at most $\left(1 + \frac{1}{2\ln 2}\right) \cdot \frac{1}{w}$. By Theorem 5 together with the lower bound established in [14], we obtain

$$2.285 \approx \tfrac{16}{7} \leq \alpha_{\text{CACTUS}} \leq 2 \cdot \left(1 + \tfrac{1}{2\ln 2}\right) \leq 3.442.$$

5 Unicyclic Graphs

Let $w \in \mathbb{N}$ and let G be a unicyclic graph with central cycle C of even length. For each $u \in V(C)$, let $T(u)$ denote the tree attached to v_j. See the following figure for an illustration:



Algorithm 3 returns a (random) subset of edges $F \in \mathcal{F}_{2w}(G)$ such that $\Pr[e \in F] \leq \frac{2}{2w-1}$ for all $e \in E(G)$.

The algorithm involves two sources of randomness. First, the algorithm selects an edge $e \in E(C)$ uniformly at random and temporarily removes it, transforming the cycle C into a path $P := C \setminus e$. The path P is then extended at both ends, which we refer to as the extended path P'. Next, the algorithm selects a sequence of roots R along P' such that consecutive roots are spaced at distance exactly $2w-1$ from each other. For each pair of consecutive roots $r_1, r_2 \in R$, the algorithm removes the middle edge of the subpath $P'[r_1, r_2]$. Then, for each vertex $u \in V(P)$, it removes the set $F_w^{l(u)}(T(u), u)$ from the tree $T(u)$ attached at u, where $0 \leq l(u) \leq w-1$ is chosen appropriately. Finally, if the algorithm ends up removing an additional edge from P (other than the initially chosen edge e), then e is permanently removed. Otherwise, if no other edge from P is removed, the edge e is retained.

Algorithm 3 Unicyclic Algorithm.

1: Let $F = \emptyset$.
2: Choose an edge $e \in E(C)$ uniformly at random. Denote the path $C \setminus e$ as P, and let $H = G \setminus e$ (see Figure 2).
3: Let P_{left}, P_{right} be two new paths, each of length $2w-1$. Construct a new graph G' by gluing P_{left} to one endpoint of P and P_{right} to the other endpoint of P in the graph H. Define the extended path $P' = P_{left} \oplus P \oplus P_{right} = (v_0, \ldots, v_k)$, where v_0, v_k are the left endpoint and the right endpoint of P', respectively. (see Figure 3)
▷ Note that $P_{left} = (v_0, \ldots, v_{2w-1})$.
4: Pick $\rho \in \{1, \ldots, 2w-1\}$ uniformly at random.
5: Let $R = \{u \in V(P') \mid d_{G'}(v_\rho, u) = 0 \mod 2w-1\}$ denote the set of roots.
▷ Every two consecutive roots in R are at distance $2w-1$ of each other.
6: **for** each two consecutive $r_1, r_2 \in R$ **do**
7: Add the middle edge of $P'[r_1, r_2]$ to F, and remove it from P'.
8: **end for**
▷ Since all edges of P' are cut-edges in G', then each connected component of $G' \setminus F$, except the ones containing the endpoints of P', has a unique root at this stage of the algorithm.
9: **for** $u \in V(P)$ **do**
10: Let $r(u) \in R$ be the unique root within the same connected component as u in $G' \setminus F$. Let $l(u) = w - 1 - d_{G'}(r(u), u)$, and add $F_w^{l(u)}(T(u), u)$ to F, and remove $F_w^{l(u)}(T(u), u)$ from $T(u)$.
▷ One can verify that $0 \leq d_{G'}(r(u), u) \leq w-1$, which implies $0 \leq l(u) \leq w-1$. So, the set $F_w^{l(u)}(T(u), u)$ is well-defined. See Remark 15.
11: **end for**
12: **if** the endpoints of e are within different connected component in $G' \setminus F$ **then**
13: Add e to F, and remove e from G.
14: **end if**
15: $F \to F \setminus (E(P_{left}) \cup E(P_{right}))$
16: Return F.

Let F' denote the set of edges in F immediately before the execution of the line 12 of the Algorithm 3.

▶ Remark 15. Each two consecutive roots in R are at distance $2w-1$ of each other, and the middle edge of the path between each two consecutive roots is contained in F'. This means that on P', any two consecutive edges in F' are at distance $2w-2$ of each other, meaning there are exactly $2w-2$ number of edges of P' between them. Since $len(P_{left}), len(P_{right}) = 2w-1$,

then $|F' \cap E(P_{left})| = |F' \cap E(P_{right})| = 1$. This means that any vertex $v \in V(P)$ is between two consecutive edges in $F' \cap E(P')$. Moreover, there are exactly $2w-2$ edges of the path P' between these two edges, which creates the path Q. The middle vertex of this sub-path Q of P' is the unique root $r(v) \in R$ denoted in the line 10 of the Algorithm 3. So, it can be derived that $0 \leq d_{G'(r(v),v)} \leq w-1$. Also, since each two consecutive roots of R are at distance $2w-1$ of each other, then $r(v)$ is the unique closest root to v in G'.

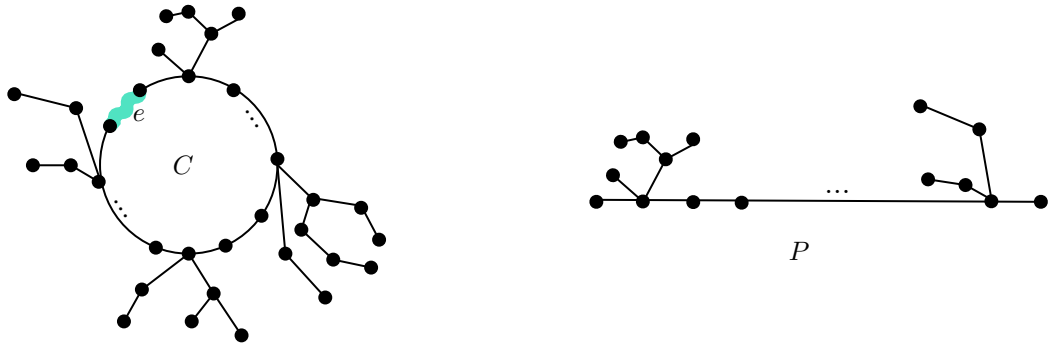


Figure 2 The left figure shows the random chosen edge $e \in E(C)$, and the right figure shows the graph $H = G \setminus e$.

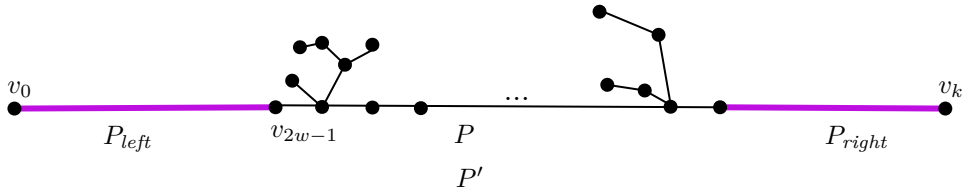


Figure 3 The extended graph G' and the extended path $P' = P_{left} \oplus P \oplus P_{right}$.

▶ **Lemma 16.** *$F \in F_{2w}(G)$.*

Proof. It suffices to prove $F' \in \mathcal{F}_{2w}(G')$: adding (or not adding) e cannot increase component diameters, and removing P_{left} and P_{right} affects only auxiliary edges outside $E(G)$.

▷ Claim 17. If $F' \in \mathcal{F}_{2w}(G')$, then $F \in \mathcal{F}_{2w}(G)$.

Proof. Assume $F' \in \mathcal{F}_{2w}(G')$. Note that $G' + e$ is a super graph of G, i.e., G is an induced sub-graph of $G + e$. So, we have $F' \cup \{e\} \in \mathcal{F}_{2w}(G' + e)$. Based on the if condition in the line 12 of the Algorithm 3, there are two cases to consider:

1. The endpoints of e are within different connected component in $G' \setminus F'$; then the line 13 is executed by the Algorithm 3, which means that $F = F' \cap \{e\}$ right after the execution of the line 14.
2. The endpoints of e are within the same connected component in $G' \setminus F'$; then the line 13 is not executed by the Algorithm 3, which means that $F = F'$ right after the execution of the line 14, and $F' \in \mathcal{F}_{2w}(G' + e)$.

Thus, in both cases, $F \in \mathcal{F}_{2w}(G')$ right after the execution of the line 14. In the line 15 the edges of P_{left}, P_{right} are being excluded from F. Thus, $F \in \mathcal{F}_{2w}(G)$ after the execution of the Algorithm 3. ◁

We prove $F' \in \mathcal{F}_{2w}(G')$ in the remaining. As stated in the Remark 15, $|F' \cap E(P_{left})| = |F' \cap E(P_{right})| = 1$, which means that the connected components containing the endpoints of P' in $G' \setminus F'$ are simple paths with length at most $2w-2$. Now, consider the other connected components in $G' \setminus F'$. Let D be a connected component not containing the endpoints of P'. If D is a subgraph of $T(u)$ for some $u \in V(C)$, then $diam(D) \leq 2w-1$, since $F_w^{l(u)}(T(u), u) \subseteq F'$, and $F_w^{l(u)}(T(u), u)$ is a $2w$-diameter decomposition for $T(u)$, as stated in the Definition 8.
Thus, assume that D is not a sub-graph of any $T(u)$. This means that $V(D) \cap V(P') \neq \varnothing$. Since $F' \cap E(P')$ must consists of edges along P' spaced exactly $2w-2$ apart using Remark 15, and D does not contain the endpoints of P', then the sub-graph induced by D on the vertices $V(P')$, denoted $D[V(P')]$, forms a contiguous sub-path Q of P' of length $2w-2$. Moreover, the unique middle vertex of Q is a root $r' \in R$. Note that the only vertex in $V(D)$ that belongs to the root set R is this single, central vertex r'.

▷ Claim 18. $d_{G'}(r', u) \leq w-1$ for all $u \in V(D)$.

Proof. If $u \in V(Q)$, then since $len(Q) = 2w-2$, and r' is the middle vertex of Q, then $d_{G'}(r', u) \leq w-1$. So, assume $u \notin V(Q)$. This means that there exists a vertex $v \in V(Q)$ such that $u \in V(T_v)$. We have $l(v) = w-1-d_{G'}(r', v)$. As stated in the Definition 8, $rad_{F_w^{l(v)}(T(u),u)}(v) \leq l(v)$, which means that $d_{G'}(v, u) \leq l(v)$. The triangle inequality implies that

$$d_{G'}(r', u) \leq d_{G'}(r', v) + d_{G'}(v, u) \leq w-1-l(v)+l(v) = w-1. \qquad \triangleleft$$

This shows that $diam(D) \leq 2w-2$. Thus, $F' \in \mathcal{F}_{2w}(G')$. ◀

Now, we are going to show that each edge $e \in E(G)$ is removed by the Algorithm 3 with probability at most $\frac{2}{2w-1}$.

▶ **Definition 19.** *For any $z \in \mathbb{Z}$, let $m_w(z) \in \{-(w-1), \ldots, 0, \ldots, w-1\}$ be the unique integer such that $z = m_w(z) \mod 2w-1$. Note that $0 \leq |m_w(z)| \leq w-1$, and $m_w(-z) = -m_w(z)$.*

▶ **Lemma 20.** *Fix the random edge $e \in E(C)$ in the line 2, and let G' and $P_{left} = (v_0, \ldots, v_{2w-1})$ be as stated in the line 3 and ρ stated in the line 4 of the Algorithm 3. For any $v \in V(C) = V(P)$, we have $d_{G'}(r(v), v) = |m_w(d_{G'}(v_0, v) - \rho)|$.*

Proof. We have $d_{G'}(v_0, r_v) = d_{G'}(v_0, v_\rho) + d_{G'}(v_\rho, r(v)) = \rho + d_{G'}(v_\rho, r(v))$, which implies $d_{G'}(v_0, r_v) = \rho \mod 2w-1$. There are two cases to consider:

1. $r(v) \in V(P'[v_0, v])$; this means that

   $$d_{G'}(r(v), v) = d_{G'}(v_0, v) - d_{G'}(v_0, r(v)),$$

   which implies

   $$d_{G'}(r(v), v) = d_{G'}(v_0, v) - \rho \mod 2w-1.$$

   Since $0 \leq d_{G'}(r(v), v) \leq w-1$, then $0 \leq m_w(d_{G'}(v_0, v) - \rho)$, which implies

   $$d_{G'}(r(v), v) = m_w(d_{G'}(v_0, v) - \rho) = |m_w(d_{G'}(v_0, v) - \rho)|.$$

2. $v \in V(P'[v_0, r(v)])$; this means that

   $$d_{G'}(r(v), v) = d_{G'}(v_0, r(v)) - d_{G'}(v_0, v),$$

which implies

$$d_{G'}(r(v), v) = -(d_{G'}(v_0, v) - \rho) \mod 2w - 1.$$

Since $0 \leq d_{G'}(r(v), v) \leq w - 1$, then $0 \leq m_w(-(d_{G'}(v_0, v) - \rho)) = -m_w(d_{G'}(v_0, v) - \rho)$, which implies

$$d_{G'}(r(v), v) = -m_w(d_{G'}(v_0, v) - \rho) = |m_w(d_{G'}(v_0, v) - \rho)|. \quad \blacktriangleleft$$

▶ Remark 21. The choice of the edge e in the line 2 is independent from the choice of ρ in the line 4 of the Algorithm 3.

▶ **Lemma 22.** *Let $u \in V(C) = V(P)$ be an arbitrary vertex, and let $k \in \{0, \ldots, w-1\}$ be an arbitrary number.*

$$\Pr[d_{G'}(r(u), u) = k] = \begin{cases} \frac{1}{2w-1} & \text{if} \quad k = 0 \\ \frac{2}{2w-1} & \text{if} \quad k = 1, \ldots, w-1. \end{cases}$$

Proof. Fix k. Let e be the random edge denoted in the line 2 in the Algorithm 3. By conditional probability, we have

$$\Pr[d_{G'}(r(u), u) = k] = \sum_{f \in E(C)} \Pr[d_{G'}(r(u), u) = k \mid e = f] \Pr[e = f].$$

$\Pr[e = f] = \frac{1}{len(C)}$ for all $f \in E(C)$. Now, to compute $\Pr[d_{G'}(r(u), u) = k \mid e = f]$, we fix $e = f$. Lemma 20 implies that $d_{G'}(r(u), u) = |m_w(d_{G'}(v_0, v) - \rho)|$, where ρ is the random number denoted in the line 4 of the Algorithm 3. Note that $d_{G'}(v_0, v)$ is a fixed value assuming $e = f$ is fixed. If $k = 0$, then $d_{G'}(r(u), u) = 0$ iff $m_w(d_{G'}(v_0, v) - \rho) = 0$, and this is equivalent to $d_{G'}(v_0, v) = \rho \mod 2w - 1$. It can be derived that $\Pr[\rho = d_{G'}(v_0, v) \mod 2w - 1] = \frac{1}{2w-1}$. Thus, $\Pr[d_{G'}(r(u), u) = 0 \mid e = f] = \frac{1}{2w-1}$. If $k \in \{1, \ldots, w-1\}$, then $d_{G'}(r(u), u) = k$ iff $m_w(d_{G'}(v_0, v) - \rho) \in \{k, -k\}$. It can be derived that the events $m_w(d_{G'}(v_0, v) - \rho) = k$ and $m_w(d_{G'}(v_0, v) - \rho) = -k$ are disjoint from each other, and each has probability $\frac{1}{2w-1}$. Thus, $\Pr[d_{G'}(r(u), u) = k \mid e = f] = \frac{2}{2w-1}$. This completes the proof. ◀

▶ **Lemma 23.** *Let $f \in E(G) \backslash E(C)$, i.e., $f \in E(T(u))$ for some $u \in V(u)$.* $\Pr[f \in F] \leq \frac{2}{2w-1}$.

Proof. As stated in the Definition 8, $\{F_k^i(T(u), u)\}_{i=0}^{k-1}$ is a partition of $E(T(u))$. Assume $f \in F_w^j(T(u), u)$ for some $j = 0, \ldots, w-1$. Line 10 implies that $f \in F$ iff $j = l(u) = w - 1 - d_{G'}(r(u), u)$. So, $e \in F$ iff $d_{G'}(r(u), u) = w - 1 - j$. Lemma 22 implies that

$$\Pr[d_{G'}(r(u), u) = w - 1 - j] = \begin{cases} \frac{1}{2w-1} & \text{if} \quad j = w - 1 \\ \frac{2}{2w-1} & \text{if} \quad j = 0, \ldots, w-2. \end{cases} \quad \blacktriangleleft$$

▶ **Lemma 24.** *Let $f \in E(C)$ be an arbitrary edge.* $\Pr[f \in F] \leq \frac{2}{2w-1}$.

Proof. Let A denote the event that $e = f$, where e is the random edge denoted in the line 2 of the Algorithm 3. Denote $\bar{A}$ as the complement of A. We compute $\Pr[f \in F]$ by conditioning on A. We have

$$\Pr[f \in F] = \Pr[f \in F|A]P[A] + \Pr[f \in F|\bar{A}]P[\bar{A}].$$

We know that $\Pr[A] = \frac{1}{len(C)}$ and $\Pr[\bar{A}] = \frac{len(C)-1}{len(C)}$. Now, if A happens, meaning that $e = f$, then f is added to the set F' in the line 13 iff the end-points of $e = f$ are in different connected components in $G' \setminus F'$. Note that the endpoints of $e = f$ are indeed the endpoints of the path P. Since G' is a tree, then $e = f$ is added to F' iff $F' \cap E(P') \neq \emptyset$. We have two cases to consider:

1. $len(C) \geq 2w$; then $len(P) \geq 2w - 1$, which means that $F' \cap E(P') \neq \varnothing$ by Remark 15. Thus, $\Pr[f \in F|A] = 1 \leq \frac{len(C)-1}{2w-1}$ in this case.
2. $len(C) < 2w$; Remark 15 implies that each edge of P' is contained in F' with probability $\frac{1}{2w-1}$, assuming $e = f$ is fixed. Using union bound, we have

$$\Pr[F' \cap E(P) \neq \varnothing] \leq \sum_{f' \in E(P)} \Pr[f' \in F'] = \frac{len(C) - 1}{2w - 1}.$$

Thus, $\Pr[f \in F|A] = \frac{len(C)-1}{2w-1}$.

Now, we compute $\Pr[f \in F|\bar{A}]$. Assuming $\bar{A}$ means $f \in E(P)$, and as mentioned before each edge of P' is present in F' with probability $\frac{1}{2w-1}$ assuming $e \neq f$ is fixed. So, $\Pr[f \in F|\bar{A}] = \frac{1}{2w-1}$. Thus,

$$\Pr[f \in F] = \frac{len(C) - 1}{2w - 1} \times \frac{1}{len(C)} + \frac{1}{2w - 1} \times \frac{len(C) - 1}{len(C)} \leq \frac{2}{2w - 1}.$$ ◀

▶ **Corollary 25.** *For the family UNICYCLIC, we have* $\alpha_{UNICYCLIC} \leq 2$.

Proof. Fix $w' \in \mathbb{N}$. By Algorithm 3, for any unicyclic graph G each edge is removed with probability at most $\frac{2}{2w'-1}$, and the algorithm outputs a random $F \in \mathcal{F}_{2w'}(G)$. Hence there exists an $SDD(G, 2w', \frac{2}{2w'-1})$.

Now fix $k \in \mathbb{N}$ and let $w' = 2kw$ for arbitrary $w \in \mathbb{N}$. Then $SDD\left(G, 4kw, \frac{2}{4kw-1}\right)$ exists for every unicyclic G. Since $\frac{2}{4kw-1} \leq \frac{2 \cdot \frac{4k}{4k-1}}{4kw}$, we also have $SDD\left(G, 4kw, \frac{2 \cdot \frac{4k}{4k-1}}{4kw}\right)$. Lemma 6 implies that for every unicyclic G and $\forall w \in \mathbb{N}$ there exists $SDD\left(G, 2w, \frac{2 \cdot \frac{4k}{4k-1}}{2w}\right)$. Applying Theorem 5 yields $\alpha_{\text{UNICYCLIC}} \leq 2 \cdot \frac{4k}{4k-1}$. Letting $k \to \infty$ gives $\alpha_{\text{UNICYCLIC}} \leq 2$. ◀

6 Path Cactus

In this section, we outline the algorithm for the path cactus. Due to space constraints, we omit the analysis here and defer it to the section 2.8 of the first author's Master's thesis [12]. We first recall the definition of a path cactus graph.

▶ **Definition 26.** *Let G be a cactus, we say that G is a path cactus if it is 2-edge connected and each cycle shares a common vertex with at most 2 more cycles, or equivalently it is a path of consecutive cycles. (Figure 4)*

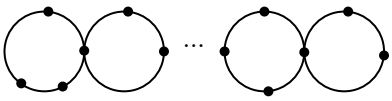


Figure 4 Path Cactus.

▶ **Definition 27.** *Let $P = (v_1, \ldots, v_n)$ be an even-length path. We denote the mid-vertex of P by $v_{\lceil \frac{n+1}{2} \rceil}$.*

▶ **Definition 28.** *Assume $n \geq 3$. Let $P = (v_1, \ldots, v_n)$ be a path. Let $e_1 = (v_i, v_{i+1}), e_2 = (v_j, v_{j+1}) \in E(P)$ be two distinct edges such that $i < j$. The distance between e_1, e_2 in P is denoted as $d_P(e_1, e_2) = j - i - 1$. Note that e_1, e_2 are consecutive edges in P iff $j = i + 1$.*

▶ **Definition 29.** *Let $C = (v_1, \ldots, v_n, v_1)$ be an even-length cycle. For any vertex v_i, we denote the opposite vertex of v_i in C as*

$$\begin{cases} v_{i+\frac{n}{2}}, & \text{if } i \leq \frac{n}{2} \\ v_{i-\frac{n}{2}}, & \text{otherwise} \end{cases}$$

Let G be a path cactus with consecutive cycles $C_1, \ldots, C_k$, such that $V(C_i) \cap V(C_{i+1}) = \{u_i\}$ for $i = 1, \ldots, k-1$. Let $u_0 \neq u_1$ be an arbitrary vertex in C_1, and also $u_k \neq u_{k-1}$ an arbitrary vertex in C_k. As mentioned before, we can assume that length of all cycles are even. There are exactly 2 simple paths between each pair of u_{i-1}, u_i in the cycle C_i for $i = 1, \ldots, k$. Let S_i denotes the shorter, and L_i denotes the longer one. (Figure 5)

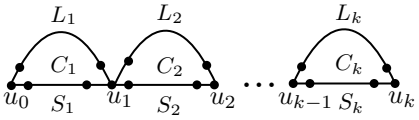


Figure 5 S_i and L_i.

Let $P = S_1 \oplus \cdots \oplus S_k$ with endpoints u_0, u_k. We refer $v_1 = u_0, \ldots, v_n = u_k$ as the ordered vertices of P. To perform the Algorithm 4, we need to consider a super graph of G. Let $P_{left} = (v_{-2w}, \ldots, v_0), P_{right} = (v_{n+1}, \ldots, v_{n+2w})$ be 2 new paths. Extend G to G' by adding an edge (v_0, v_1), and another edge (v_n, v_{n+1}). Let $P' = P_{left} \oplus (v_0, v_1) \oplus P \oplus (v_n, v_{n+1}) \oplus P_{right}$. (Figure 6)

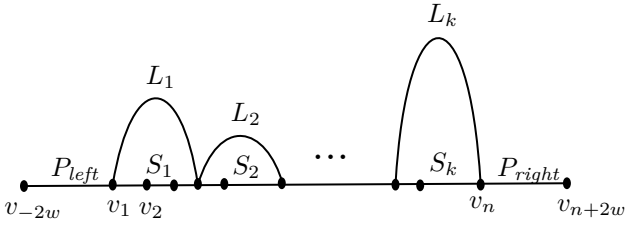


Figure 6 G' and P'.

▶ Remark 30. $d_{G'}(u, v) = d_G(u, v)$ for any two vertives $u, v \in V(G)$. Moreover, $u_0, u_1, \ldots, u_k$ are cut-vertices of G'.

Given $w \in \mathbb{N}$, Algorithm 4 selects "blue" edges on P' so that consecutive blue edges are spaced at distance $2w - 1$. Based on these blue edges, it removes additional edges from G'. We have the following theorem (proof omitted here due to space; see the thesis [13]):

▶ **Theorem 31.** *Let $w \in \mathbb{N}$, let G be a path–cactus graph in which every cycle has even length, and let G' be the extended graph used by Algorithm 4. Then Algorithm 4 outputs a random set $F' \in \mathcal{F}_{2w}(G')$ such that*

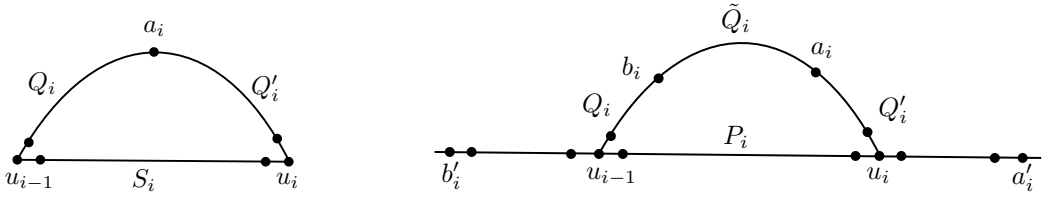
$$\Pr[e \in F'] \leq \tfrac{1}{w} \qquad \text{for all } e \in E(G').$$

Hence $SDD(G', 2w, \frac{1}{w})$ exists. Since G is a subgraph of G', the same diameter and marginal bounds hold for edges of G, and therefore $SDD(G, 2w, \frac{1}{w})$ exists as well.

To describe Algorithm 4, we first introduce the following definitions.

▶ **Definition 32.** *We say that a cycle C_i is long if $len(L_i) > 2w - 1$. For each long cycle C_i, let a_i be the mid-vertex of L_i. Let $Q_i = L_i[u_{i-1}, a_i], Q'_i = L_i[a_i, u_i]$. Note that $len(Q_i) = len(Q'_i) \geq w$. (Figure 7a)*

▶ **Definition 33.** *We say that a cycle C_i is short if $len(L_i) \leq 2w - 1$. For each short cycle C_i, let a_i, b_i be the opposite points of u_{i-1}, u_i in C_i, respectively. Let $Q_i = L_i[u_{i-1}, a_i], Q'_i = L_i[u_i, b_i]$. Also, let $\tilde{Q}_i = L_i[b_i, a_i]$ be the common sub-path of Q_i, Q'_i. Let C_i be a short cycle. Let $a'_i \in V(P'[u_i, v_{n+2w}])$ be the unique vertex with $d(u_{i-1}, a'_i) = \frac{len(C_i)}{2}$. Similarly, let $b'_i \in V(P'[v_{-2w}, u_{i-1}])$ be the unique vertex with $d(u_i, b'_i) = \frac{len(C_i)}{2}$. Moreover, let $P_i = P'[a'_i, b'_i]$. (Figure 7b)*



(a) Long cycle. **(b)** Short cycle.

Figure 7 Path-cactus illustrations: (a) long cycle; (b) short cycle.

Now, we are ready to describe the Algorithm 4.

Algorithm 4 Path Cactus Algorithm.

Pick uniformly random edges from P' such that the distance of every consecutive edges is $2w - 1$, and call them blue edges.
for each long cycle C_i **do**
    Let o_i be the distance of the closest blue edge to u_{i-1}. Pick edges with distance $o_i + k'w(k' = 0, \ldots, \lfloor \frac{len(Q_i) - o_i}{w} \rfloor)$ from u_{i-1} in Q_i.
    Let z_i be the distance of the closest blue edge to u_i. Pick edges with distance $z_i + k'w(k' = 0, \ldots, \lfloor \frac{len(Q'_i) - z_i}{w} \rfloor)$ from u_i in Q'_i.
    Remove all the picked edges from C_i.
end for
for each short cycle C_i **do**
    if P_i contains a blue edge, **then**
        Let f_i be the blue edge in P_i.
        Let $o_i = d_{P'}(u_{i-1}, f_i)$. Pick the unique edge f with $d_{Q_i}(u_{i-1}, f) = o_i$ in Q_i.
        Let $z_i = d_{P'}(u_i, f_i)$. Pick the unique edge f' with $d_{Q'_i}(u_i, f') = z_i$ in Q'_i.
        Remove f, f' from C_i.
    end if
end for
Remove the blue edges.

References

1 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM Journal on Computing*, 48(3):1120–1145, 2019. doi:10.1137/17M1112406.

2 Cédric Bentz. Edge disjoint paths and max integral multiflow/min multicut theorems in planar graphs. *Electronic Notes in Discrete Mathematics*, 22:55–60, 2005. doi:10.1016/J.ENDM.2005.06.010.

3 Jonathan Conroy and Arnold Filtser. How to protect yourself from threatening skeletons: Optimal padded decompositions for minor-free graphs. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 2281–2292, 2025. doi:10.1145/3717823.3718252.

4 R Diestel. Graph theory (vol. 173), 2010.

5 Jittat Fakcharoenphol and Kunal Talwar. An improved decomposition theorem for graphs excluding a fixed minor. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 36–46. Springer, 2003. doi:10.1007/978-3-540-45198-3_4.

6 Arnold Filtser, Tobias Friedrich, Davis Issac, Nikhil Kumar, Hung Le, Nadym Mallek, and Ziena Zeif. Optimal padded decomposition for bounded treewidth graphs. *arXiv preprint arXiv:2407.12230*, 2024. doi:10.48550/arXiv.2407.12230.

7 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

8 Tobias Friedrich, Davis Issac, Nikhil Kumar, Nadym Mallek, and Ziena Zeif. Approximate max-flow min-multicut theorem for graphs of bounded treewidth. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1325–1334, 2023. doi:10.1145/3564246.3585150.

9 Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996. doi:10.1137/S0097539793243016.

10 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. doi:10.1007/BF02523685.

11 T Chiang Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963.

12 Sina Kalantarzadeh. Cuts in optimization and approximation: Generalized MIR cutting plane and integrality gap bounds for the planar multicut problem. Master's thesis, University of Waterloo, 2025.

13 Sina Kalantarzadeh. *Cuts in Optimization and Approximation: Generalized MIR Cutting Plane and Integrality Gap Bounds for the Planar Multicut Problem*. Master's thesis, University of Waterloo, 2025. URL: https://hdl.handle.net/10012/22484.

14 Sina Kalantarzadeh and Nikhil Kumar. Improved lower bounds on multiflow-multicut gaps, 2025. doi:10.48550/arXiv.2507.06576.

15 Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 682–690. ACM, 1993. doi:10.1145/167088.167261.

16 Eva Tardos and Vijay V Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs. *Information Processing Letters*, 47(2):77–80, 1993. doi:10.1016/0020-0190(93)90228-2.

A $\frac{3}{2}$ Lower Bound for CYCLE

In the following, Lemma 34 shows that $\alpha_{\text{CYCLE}} \geq \frac{3}{2} - \frac{2}{2k+1}$ for any $k \in \mathbb{N}$. Letting $k \to \infty$ implies $\alpha_{\text{CYCLE}} \geq \frac{3}{2}$.

▶ **Lemma 34.** *Let M be the following minimum multicut instance. The base graph is a cycle C with $2k+1$ vertices and unit edge costs $c_e = 1$ for all $e \in E(C)$. The source–sink pairs are $\{(u,v) \mid d_C(u,v) = k\}$. Then*

$$\frac{OPT_{IP}(M)}{OPT_{LP}(M)} \geq \frac{3}{2} - \frac{2}{2k+1}.$$

Proof. We claim $\mathrm{OPT}_{IP}(M) = 3$. Indeed, removing any two edges from C yields two path components whose total length is $(2k+1) - 2 = 2k - 1$. Hence one component has length at least k. That component contains a subpath of length exactly k, which separates some designated source–sink pair at distance k; thus two edges never suffice. Conversely, deleting three suitably placed edges clearly disconnects all such pairs, so $\mathrm{OPT}_{IP}(M) = 3$.
For the LP relaxation (1), set $x_e = \frac{1}{k}$ for all $e \in E(C)$. Every designated s–t path has exactly k edges, so $x(P) = \sum_{e \in P} x_e = k \cdot \frac{1}{k} = 1$, and x is feasible. Therefore

$$\mathrm{OPT}_{LP}(M) \leq \sum_{e \in E(C)} c_e x_e = (2k+1) \cdot \tfrac{1}{k} = \frac{2k+1}{k}.$$

It follows that

$$\frac{\mathrm{OPT}_{IP}(M)}{\mathrm{OPT}_{LP}(M)} \geq \frac{3}{(2k+1)/k} = \frac{3k}{2k+1} = \frac{3}{2} - \frac{2}{2k+1}.$$ ◀

B Proof of Lemma 6

Proof. Fix $G \in \mathcal{G}$ and $w \in \mathbb{N}$. Create G' from G by subdividing every edge of G into a path of length $2k$. Since $\mathcal{G}$ is closed under subdivisions, $G' \in \mathcal{G}$; moreover, every cycle length in G' is multiplied by $2k$, hence divisible by $2k$, so $G' \in \mathcal{G}'$. By assumption, there exists an SDD distribution $\mathcal{D}' = \{y'_{F'}\}_{F' \in \mathcal{F}_{4kw}(G')}$ with

$$\Pr_{F' \sim \mathcal{D}'}[e' \in F'] \leq \tfrac{\alpha}{4kw} \qquad \text{for every } e' \in E(G').$$

Coarsening map. For $F' \subseteq E(G')$, define the coarsening

$$\Phi(F') = \{\, e \in E(G) \;:\; \text{at least one subdivision edge of } e \text{ lies in } F' \,\}.$$

We claim: if F' is a $(4kw)$-diameter decomposition of G', then $\Phi(F')$ is a $(2w)$-diameter decomposition of G. Indeed, distances in G' are exactly $2k$ times the corresponding distances in G (every original edge became a path of length $2k$). Removing F' leaves every component of G' with diameter $< 4kw$ (in the G' metric), which translates to diameter $< (4kw)/(2k) = 2w$ in the G metric after contracting each subdivided path back to its original edge. Thus $\Phi(F') \in \mathcal{F}_{2w}(G)$. Define a distribution $\mathcal{D} = \{y_F\}_{F \in \mathcal{F}_{2w}(G)}$ by

$$y_F = \sum_{\substack{F' \in \mathcal{F}_{4kw}(G') \\ \Phi(F') = F}} y'_{F'}.$$

This is a valid probability distribution since the Φ-preimages of distinct F's are disjoint and $\sum_{F'} y'_{F'} = 1$. Fix $e \in E(G)$, and let its $2k$ subdivision edges in G' be $e'_1, \ldots, e'_{2k}$. Then

$$\Pr_{F \sim \mathcal{D}}[e \in F] = \Pr_{F' \sim \mathcal{D}'}[\Phi(F') \ni e] = \Pr_{F' \sim \mathcal{D}'}\Big[\bigvee_{i=1}^{2k} (e'_i \in F')\Big]$$
$$\leq \sum_{i=1}^{2k} \Pr_{F' \sim \mathcal{D}'}[e'_i \in F'] \leq 2k \cdot \tfrac{\alpha}{4kw} = \tfrac{\alpha}{2w},$$

where we used the union bound and the marginal bound in $\mathcal{D}'$. Therefore $\mathcal{D}$ witnesses $SDD(G, 2w, \alpha/(2w))$, as required. ◀

C 2 Lower Bound for PATH-CACTUS

We give a path–cactus instance with integrality gap approaching 2. Let the graph consist of n triangles $\Delta_1, \ldots, \Delta_n$ in a path, where Δ_i has vertices u_{i-1}, u_i, v_i and consecutive triangles share u_i. Set $c(u_i, v_i) = c(u_{i-1}, v_i) = 1$ for all i, and assign arbitrarily large cost to all other edges. Let the demand pairs be all (v_i, v_j) with $1 \le i < j \le n$.

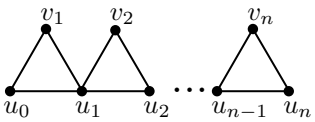


Figure 8 Path–cactus lower-bound construction.

Fractional solution: set $x(u_i, v_i) = x(u_{i-1}, v_i) = \frac{1}{2}$ for all i. Every v_i–v_j path crosses $2(j-i)$ such edges, so $x(P) \ge 1$; the LP cost is n.
Integral cost: to separate all (v_i, v_j) pairs, at least two unit-cost edges must be cut in each of at least $n-1$ triangles, hence any multicut costs $\ge 2n-2$.
Therefore the gap is

$$\frac{\mathrm{OPT}_{IP}}{\mathrm{OPT}_{LP}} \ge \frac{2n-2}{n} \xrightarrow[n\to\infty]{} 2,$$

so $\alpha_{\text{PATH-CACTUS}} \ge 2$.

Simultaneously Fair Allocation of Indivisible Items Across Multiple Dimensions

Yasushi Kawase
The University of Tokyo, Japan

Bodhayan Roy
Indian Institute of Technology Kharagpur, India

Mohammad Azharuddin Sanpui[1]
Indian Institute of Technology Kharagpur, India

Abstract

This paper explores the fair allocation of indivisible items in a multidimensional setting, motivated by the need to address fairness in complex environments where agents assess bundles according to multiple criteria. Such multidimensional settings are not merely of theoretical interest but are central to many real-world applications. For example, cloud computing resources are evaluated based on multiple criteria such as CPU cores, memory, and network bandwidth. In such cases, traditional one-dimensional fairness notions fail to capture fairness across multiple attributes. To address these challenges, we study two relaxed variants of envy-freeness: *weak simultaneously envy-free up to c goods (weak sEFc)* and *strong simultaneously envy-free up to c goods (strong sEFc)*, which accommodate the multidimensionality of agents' preferences. Under the weak notion, for every pair of agents and for each dimension, any perceived envy can be eliminated by removing, if necessary, a different set of goods from the envied agent's allocation. In contrast, the strong version requires selecting a single set of goods whose removal from the envied bundle simultaneously eliminates envy in every dimension. We provide upper and lower bounds on the relaxation parameter c that guarantee the existence of weak or strong sEFc allocations, where these bounds are independent of the total number of items. In addition, we present algorithms for checking whether a weak or strong sEFc allocation exists. Moreover, we establish NP-hardness results for checking the existence of weak sEF1 and strong sEF1 allocations.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Solution concepts in game theory

Keywords and phrases Fair allocation, Envy-free up to one good, Multi-dimensional criteria, Linear programming, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.41

Related Version *Full Version*: https://arxiv.org/abs/2506.21727

Funding *Yasushi Kawase*: Supported by JSPS KAKENHI Grant Number JP25K00137, by JST PRESTO Grant Number JPMJPR2122, by JST ERATO Grant Number JPMJER2301, and by Value Exchange Engineering, a joint research project between Mercari R4D Lab and RIISE (Research Institute for an Inclusive Society through Engineering).

1 Introduction

Fair allocation of resources is a central topic in economics, operations research, computer science, and public policy [2, 19, 28, 30]. Among the various notions of fairness studied in the literature, one of the most prominent is *envy-freeness*. An allocation is said to be envy-free if each agent weakly prefers her own bundle to that of any other agent, according to her preferences.

[1] Corresponding author

© Yasushi Kawase, Bodhayan Roy, and Mohammad Azharuddin Sanpui;
licensed under Creative Commons License CC-BY 4.0

While envy-freeness can always be achieved when resources are divisible [3, 9], it is often unattainable for indivisible resources. For example, when allocating a single valuable indivisible item to two agents, there is no envy-free complete allocation. This motivates an increasing amount of research to explore the relaxations of envy-freeness, such as *envy-freeness up to c items (EFc)* [24, 35]. An allocation is EFc if, for any pair of agents, any envy one agent feels towards another can be eliminated by removing at most c items from the latter's bundle, where c is a positive integer.

Traditionally, resource allocation models have focused on trying to be fair with respect to a single attribute, such as profit or cost [4, 24]. However, in practice, the value of an item is rarely determined by a single criterion [6, 10, 14]. For example, cloud computing resources are evaluated based on multiple criteria such as CPU cores, threads, GPU capabilities, memory, disk space, network bandwidth, energy efficiency. In e-commerce, products are evaluated by price, quality, sustainability, and user ratings. Even in everyday scenarios – such as dividing household chores or allocating office space – multiple factors must be considered. This multidimensionality raises a natural question:

> *To what extent can relaxed notions of fairness be achieved across all attributes simultaneously, and how can such a fair allocation be computed efficiently?*

In this paper, we investigate the fair allocation of indivisible items in a multidimensional setting, where each item is characterized by multiple attributes and agents have attribute-dependent preferences.

Our problem is closely related to a model recently introduced by Bu et al. [10], who consider a fair resource allocation framework that accounts for the preferences of both the allocator and the agents. Their model can be viewed as a special case of our setting, where each item has two attributes – one according to the allocator and one according to the agents. They introduced a fairness criterion called *doubly EFc*: an allocation is called doubly EFc if it is EFc for the agents and also EFc from the allocator's perspective. They provided the existence of a doubly EF1 allocation when there are two agents. However, they showed that a triply EF1 allocation (i.e., EF1 from three standpoints) may not exist.

In this work, we introduce and analyze the notion of *simultaneous EFc*, which extends the concept of doubly EFc to settings with an arbitrary number of attributes, rather than being limited to two or three. In the setting of Bu et al. [10], the set of items removed to eliminate envy can be chosen separately for each attribute, reflecting the perspective of different stakeholders for each attribute. However, such flexibility may not always be appropriate in our multi-attribute context. In many real-world scenarios, the attributes represent different criteria or dimensions evaluated by the same agents, rather than by distinct individuals depending on the attribute. In such cases, it is more natural and meaningful to require that the set of items removed to mitigate envy be the same across all attributes. Motivated by this observation, we define two variants of simultaneous EFc: one that allows attribute-specific removals and another that requires a common removal set for all attributes.

1.1 Our Results

We study the fair allocation of indivisible items in a multidimensional setting, where each agent's valuation is represented by an ℓ-dimensional vector. In this framework, we explore two notions of relaxed envy-freeness: (i) *weak simultaneously envy-free up to c goods (weak sEFc)*, and (ii) *strong simultaneously envy-free up to c goods (strong sEFc)*. Notably, weak sEFc in the two-dimensional case corresponds to the doubly EFc and was introduced by Bu et al. [10]. In contrast, strong sEFc is a new fairness concept introduced in this paper.

Under weak sEFc, for any pair of distinct agents i and i', and for every evaluation dimension, any envy that agent i may have toward agent i' can be eliminated by removing at most c items from agent i''s allocation in that dimension. In contrast, strong sEFc requires that, for every pair of agents i and i', there exists a set of at most c items whose removal from agent i''s bundle simultaneously eliminates the envy of agent i in every dimension.

Bu et al. [10] provided an instance with two identical agents and three dimensions that admits no weak sEF1 allocation (Theorem 3). We observe that this result can be extended to the general ℓ-dimensional case by showing that $c = \Omega(\sqrt{\ell})$ is a lower bound to guarantee the existence of a weak sEFc allocation (Theorem 5). For the strong version, we establish a stronger lower bound: namely, that $c = \lceil \ell/2 \rceil$ is necessary to guarantee the existence of a strong sEFc allocation (Theorem 6).

Next, we show that when there are two agents, a strong sEF$(2\ell - 1)$ allocation always exists and can be computed in polynomial time (Theorem 7). Furthermore, for a general number of agents, we prove the existence of a strong sEF$(n^2\ell^2)$ allocation (Theorem 9). We remark that since every strong sEFc allocation is also weak sEFc by definition, these existence results immediately apply to the weak case as well. In addition, we develop algorithms to decide whether a weak or strong sEFc allocation exists in time $(mv_{\max})^{O(cn^2\ell)}$, where $v_{\max}$ denotes the maximum value assigned by any agent to any item in any dimension (Theorem 10 and Theorem 11).

Finally, we complement our existence results with computational intractability findings. Specifically, we prove that even checking for the existence of a weak sEF1 allocation is NP-complete, even when there are only two identical agents with binary valuations (Theorem 12). Moreover, we demonstrate that the corresponding decision problem for strong sEF1 allocations is NP-complete even in the case of two identical agents and two dimensions (Theorem 13). Further, we demonstrate that determining whether a given allocation is strong sEFc is NP-complete even for the identical binary case, when c is a parameter.

1.2 Related Work

Barman et al. [6] studied fair division with market values, a special case of the fair allocation with the allocator's preference. Specifically, their model introduces a market valuation that serves as a distinct valuation profiles shared identically by all agents. This results in a two-dimensional valuation framework that comprises agents' personalized valuations alongside a universal market valuation.

A number of studies have explored fair division among groups of agents [5, 7, 13, 15, 26, 32–34]. In these models, $\nu = \nu_1 + \cdots + \nu_n$ agents will be partitioned into n (≥ 2) groups, with group i comprising ν_i (≥ 1) agents. It is crucial to point out that when each group consists an identical number of agents (i.e., $\nu_1 = \cdots = \nu_n = \nu/n$), the fair division problem under this group framework aligns with our model, in which each group can be regarded as a single agent with ν/n perspectives. Moreover, even when the number of agents in each group are heterogeneous, we can reduce an instance of group framework to our setting with $\max_{i' \in [n]}$ dimensions by adding $\max_{i' \in [n]} \nu_{i'} - \nu_i$ dummy agents to each group $i \in [n]$ such that any item evaluates to 0.

Kyropoulou et al. [22] extended the classical EFc concept to a group setting by requiring that an agent's envy toward another group can be eliminated by removing at most c items from that group's allocation. When the number of agents in each group is the same, the definitions of group EFc coincides with our notion of weak EFc. For binary valuations, Kyropoulou et al. [22] determined the cardinalities of groups for which a group EF1 allocation always exists. Moreover, they demonstrated that determining the existence of a group EF1

allocation for two groups of agents with binary valuations is NP-complete. Note that their reduction does not extend to the setting with *identical* valuations in our setting. In contrast, we show via a different reduction that checking the existence of a weak sEF1 allocation is NP-complete even for identical binary valuations with two agents.

When there are two groups of agents, Segal-Halevi and Suksompong [33] proved that a group EF$(\nu - 1)$ allocation always exists, where $\nu = \nu_1 + \nu_2$. This result implies existence of a weak EF$(2\ell - 1)$ allocation in our setting with two agents and ℓ dimensions, although they left efficient computability. In contrast, we provide existence and efficient computability for strong EF$(2\ell - 1)$.

Using the discrepancy theory, Manurangsi and Suksompong [27] proved that a group EFc allocation with $c = O(\sqrt{\nu})$ always exists. This implies the existence of a weak EFc allocation with $c = O(\sqrt{n\ell})$ in our setting. Moreover, they showed that $c = \Omega(\sqrt{\nu})$ is necessary to guarantee the existence of group EFc allocations when there are constant number of groups. This implies that $c = \Omega(\sqrt{\ell})$ is necessary to guarantee the existence of weak EFc allocations. Furthermore, there are some recent works [11, 25] that asymptotically improve the lower bound for group fairness for group fair division, when the number of groups is super-constant.

Furthermore, our model involving multi-dimensional items share some similarities with the multi-layered cake cutting problem [12, 18, 20, 21, 23, 29]. However, unlike our setting, in the multi-layered cake cutting problem each layer of the cake at the same position must be allocated to different agents.

2 Preliminaries

In this paper, we address the problem of assigning m items, each evaluated according to ℓ different criteria, to n agents. We refer to this as the Multi-Dimensional Fair Allocation (MDFA) problem.

For a positive integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Let $N = [n]$ denote the set of agents and $M = \{g_1, g_2, \ldots, g_m\}$ denote the set of indivisible items. In addition, let $L = [\ell]$ denote the set of dimensions. Each agent $i \in N$ has a valuation function $v_i \colon M \to \mathbb{Z}_+^\ell$, which assigns to every item an ℓ-dimensional vector of nonnegative integers. We write $v_{ijk} = v_i(g_j)_k$ to indicate the value in the kth dimension of item $g_j \in M$ for agent $i \in N$. An instance of the MDFA problem is represented as $(N, M, L, (v_i)_{i \in N})$.

An allocation $\mathbf{A} = (A_1, A_2, \ldots, A_n)$ is a partition of the items M (i.e., $\bigcup_{i \in N} A_i = M$ and $A_i \cap A_{i'} = \emptyset$ for any distinct agents $i, i' \in N$), where each subset $A_i \subseteq M$ is allocated to agent $i \in N$. The total valuation of agent i for her allocated set A_i is defined as $v_i(A_i) = \sum_{g_j \in A_i} v_i(g_j)$. Specifically, the kth dimensional value of agent i for the allocated set A_i is defined as $v_i(A_i)_k = \sum_{g_j \in A_i} v_{ijk}$.

We call an instance of the MDFA problem *binary* if $v_{ijk} \in \{0, 1\}$ for every agent $i \in N$, every item $g_j \in M$, and every dimension $k \in L$. An instance is said to be *identical* if every agent has the same valuation function, i.e, $v_1 = v_2 = \cdots = v_n$.

We now introduce two fairness notions. Let c be a nonnegative integer.

▶ **Definition 1** (weak sEFc [10]). *An allocation* $\mathbf{A}$ *is said to be* weak simultaneously envy-free up to c goods (weak sEFc) *if for any distinct agents* $i, i' \in N$ *and any dimension* $k \in L$, *there exists a set* $X \subseteq A_{i'}$ *such that* $|X| \leq c$ *and* $v_i(A_i)_k \geq v_i(A_{i'} \setminus X)_k$.

▶ **Definition 2** (strong sEFc). *An allocation* $\mathbf{A}$ *is said to be* strong simultaneously envy-free up to c goods (strong sEFc) *if for any distinct agents* $i, i' \in N$, *there exists a set* $X \subseteq A_{i'}$ *such that* $|X| \leq c$ *and* $v_i(A_i)_k \geq v_i(A_{i'} \setminus X)_k$ *for any dimension* $k \in L$.

Clearly, strong sEFc implies weak sEFc, and weak sEFc implies strong sEFℓc.

Table 1 Valuations for an instance that admits no weak sEF1 allocations.

item g_j	g_1	g_2	g_3
v_{ij1}	1	1	0
v_{ij2}	1	0	1
v_{ij3}	0	1	1

Table 2 Valuations for an instance that admits no strong sEF1 allocation.

item g_j	g_1	g_2	g_3
v_{ij1}	1	2	0
v_{ij2}	1	0	2

Bu et al. [10] presented an instance in which no weak sEF1 allocation exists. The instance consists of two agents and three items, with valuations as shown in Table 1. In order to achieve EF1 for the first dimension, items g_1 and g_2 must be allocated to different agents. Similarly, considering the second dimension, items g_1 and g_3 must be allocated to different agents, and for the third dimension, items g_2 and g_3 must be allocated to different agents. However, no allocation can satisfies all these conditions simultaneously, implying that no weak sEF1 allocation exists.

▶ **Theorem 3** (Bu et al. [10]). *Even when there are two identical agents and the number of dimensions is three, a weak sEF1 allocation may not exist.*

It is known that a weak sEF1 allocation is guaranteed to exist when there are two agents and two dimensions [10]. In contrast, we now demonstrate that a strong sEF1 allocation may fail to exist even when there are two identical agents and two dimensions. To see this, consider an instance with three items, whose valuations are given in Table 2. Suppose, without loss of generality, that g_1 is allocated to agent 1. Then, in order to satisfy the EF1 condition for the first dimension, the item g_2 must be allocated to agent 2. Similarly, to achieve EF1 for the second dimension, the item g_3 must be also allocated to agent 2. Thus, the resulting allocation is $\mathbf{A} = (\{g_1\}, \{g_2, g_3\})$. However, this fails the strong sEF1 condition: Agent 1 envies agent 2 unless both of the two items are removed from agent 2's bundle. Hence, in this case, no strong sEF1 allocation exists.[2] Symmetric reasoning applies if g_1 is initially assigned to agent 2.

▶ **Theorem 4.** *Even when there are two identical agents and the number of dimensions is two, a strong sEF1 allocation may not exist.*

3 Non-Existence

In this section, we present two non-existence results that illustrate the inherent challenges in achieving fairness under the multi-dimensional framework. In particular, for every nonnegative integer c, we show that even when agents have binary and identical valuations, there exist instances with only two agents where no allocation satisfies either weak or strong sEFc.

We first establish that weak sEFc allocations may fail to exist even in binary identical instances with only two agents, when $c = \Omega(\sqrt{\ell})$. The construction of this instance is essentially equivalent to the one given by Manurangsi and Suksompong [27] for fair division among two groups of agents. The idea is based on discrepancy theory [1].

[2] The allocation satisfies weak sEF1 because envy can be eliminated by removing different items in each dimension.

▶ **Theorem 5.** *For any nonnegative integer c, there exists a binary identical instance with two agents and $O(c^2)$ dimensions that does not admit a weak sEFc allocation.*

Proof. Let r be the smallest power of two that is at least $4c^2+1$. Let $H \in \{-1,+1\}^{r\times r}$ be an Hadamard matrix of order r, i.e., a $\{-1,+1\}$-matrix whose row vectors are mutually orthogonal. Note that such a matrix can be constructed. Let J denote the all-1 matrix of order r and define $H^* = (H+J)/2$. It is known that for any $u \in \{-1,+1\}^r$, we have $\|H^*u\|_\infty \geq \sqrt{r}/2$ (see, e.g., [1, Section 13.4]).

Consider the instance defined by $N = \{1,2\}$, $M = \{g_1, g_2, \ldots, g_r\}$, and $L = [r]$. For every agent $i \in N$, item $g_j \in M$, and dimension $k \in L$, define $v_{ijk} = H^*_{jk}$. This instance is both binary and identical. Note that the number of dimensions r is at most $2(4c^2+1) = O(c^2)$.

For an allocation $\mathbf{A} = (A_1, A_2)$, consider the column vector $u \in \{-1,+1\}^r$ defined by $u_j = +1$ if $g_j \in A_1$ and $u_j = -1$ if $g_j \in A_2$. Then, we have $\max_{k\in L} |v_1(A_1)_k - v_1(A_2)_k| = \|H^*u\|_\infty \geq \sqrt{r}/2 > c$. This means that we need to remove at least $c+1$ items to eliminate envy. This completes the proof. ◀

This theorem has a direct consequence for the strong variant. Because any strong sEFc allocation must also satisfy weak sEFc, the impossibility of achieving weak sEFc under certain conditions immediately implies that strong sEFc allocations cannot exist under the same conditions. We further strengthen this conclusion by demonstrating that even when the number of dimensions ℓ is exactly $2c+1$, a strong sEFc allocation may fail to exist. Thus, requiring $\ell \leq 2c$, or equivalently $c \geq \lceil \ell/2 \rceil$, is necessary to guarantee the existence of a strong sEFc allocation.

▶ **Theorem 6.** *For any nonnegative integer c, there exists a binary instance with identical valuations for two agents and with $2c+1$ dimensions that does not admit a strong sEFc allocation.*

Proof. Consider the instance with $N = \{1,2\}$, $M = \{g_1, g_2, \ldots, g_{2c+1}\}$, and $L = [2c+1]$. For every agent $i \in N$, item $g_j \in M$, and dimension $k \in L$, define v_{ijk} to be 1 if $j = k$ and 0 otherwise. This instance is both binary and identical.

Now, suppose toward a contradiction that there exists a strong sEFc allocation $\mathbf{A} = (A_1, A_2)$. Without loss of generality, assume that $|A_1| \leq |A_2|$. Since $|A_1|+|A_2| = |M| = 2c+1$, it follows that $|A_1| \leq c$ and $|A_2| \geq c+1$.

Consider agent 1's evaluation of agent 2's bundle. By the definition of v_{ijk}, only item $g_{j^*} \in A_2$ contributes a value of 1 in dimension j^*. To eliminate envy in dimension j^*, the item g_{j^*} must be removed from A_2. Therefore, in order to eliminate envy in all dimensions, every item in A_2 must be removed. However, since $|A_2| \geq c+1$, at least $c+1$ items must be removed, exceeding the allowable limit of c.

This contradiction shows that no strong sEFc allocation exists for this instance. ◀

4 Existence and Computability

In this section, we present several positive results on the existence and efficient computability of weak and strong sEFc allocations. In particular, we show existence and polynomial-time computability of a strong sEFc allocation for a certain c that is independent of the number of items. Moreover, we develop dynamic programming based algorithms that can determine the existence of a weak or strong sEFc allocation in polynomial time when c, the number of agents n, and the number of dimensions ℓ are constants and the valuations are encoded in unary.

We first demonstrate that there exists a strong sEF$(2\ell-1)$ allocation and can be computed in polynomial time when there are two agents. Since every strong sEF allocation is also a weak sEF allocation, this result immediately implies the existence (and efficient computability) of weak sEF$(2\ell-1)$ allocations as well. Note that Goldberg et al. [17, Theorem 1] uses a similar polytope-based technique to divide the items into two equal parts for multiple agents. Using their polytope, we can obtain a strong sEF(2ℓ) allocation.

▶ **Theorem 7.** *When there are two agents and c is at least twice the number of dimensions minus one (i.e., $c \geq 2\ell-1$), a strong sEFc allocation always exists. Moreover, such an allocation can be computed in polynomial time.*

Proof. Let $(N, M, L, (v_i)_{i\in N})$ be a given instance of the MDFA problem. We assume that $N = \{1, 2\}$, $M = \{g_1, \dots, g_m\}$, and $L = [\ell]$. Consider the following linear program:

$$\begin{aligned}
\max_{x\in\mathbb{R}^m} \quad & \textstyle\sum_{j=1}^m v_{1j1}(2x_j - 1) & & \\
\text{s.t.} \quad & \textstyle\sum_{j=1}^m v_{1jk}(2x_j - 1) \geq 0 & (\forall k \in [\ell] \setminus \{1\}), & \quad (1)\\
& \textstyle\sum_{j=1}^m v_{2jk}(1 - 2x_j) \geq 0 & (\forall k \in [\ell]), & \quad (2)\\
& 0 \leq x_j \leq 1 & (\forall j \in [m]). & \quad (3)
\end{aligned}$$

A solution x with a nonnegative objective yields a simultaneously envy-free fractional allocation, where x_j is interpreted as the fraction of item g_j allocated to agent 1. Note that, since $x = (1/2, 1/2, \dots, 1/2)$ is a feasible solution, the optimal objective value is nonnegative.

Since the LP has m variables, any basic solution is determined by m linearly independent tight constraints. In particular, at least $m-(2\ell-1)$ of these tight constraints must be of the form $x_j = 0$ or $x_j = 1$ (i.e., constraints in (3)). This implies that every basic solution has at least $m - 2\ell + 1$ coordinates that are integral (i.e., equal to 0 or 1). Equivalently, there can be at most $2\ell - 1$ fractional coordinates.

We now construct an allocation from a basic optimal solution x^*. Note that a basic optimal solution must exist since the feasible region is nonempty and bounded. Define

$$\begin{aligned}
I_1 &\coloneqq \{g_j \in M : x^*_j = 1\}, & I_2 &\coloneqq \{g_j \in M : x^*_j = 0\},\\
F_1 &\coloneqq \{g_j \in M : 1/2 \leq x^*_j < 1\}, & F_2 &\coloneqq \{g_j \in M : 0 < x^*_j < 1/2\}.
\end{aligned}$$

Then, since all fractional coordinates of x^* lie in $F_1 \cup F_2$, we have $|F_1| + |F_2| \leq 2\ell - 1$.

Define the allocation $\mathbf{A} = (A_1, A_2)$ by $A_1 = I_1 \cup F_1$ and $A_2 = I_2 \cup F_2$. Since x^* satisfies (1) and the objective value is nonnegative, for every dimension $k \in [\ell]$ we have

$$\begin{aligned}
v_1(A_1)_k = \textstyle\sum_{g_j\in A_1} v_{1jk} &\geq \textstyle\sum_{g_j\in A_1} v_{1jk}(2x^*_j - 1)\\
&\geq \textstyle\sum_{g_j\in A_2} v_{1jk}(1 - 2x^*_j) \geq \sum_{g_j\in I_2} v_{1jk} = v_1(A_2 \setminus F_2)_k,
\end{aligned}$$

where the second inequality holds since, for the optimal solution x^*, the objective value of LP is nonnegative and inequalities in (1) are satisfied. Similarly, by (2), for every dimension $k \in [\ell]$ we have

$$\begin{aligned}
v_2(A_2)_k = \textstyle\sum_{g_j\in A_2} v_{2jk} &\geq \textstyle\sum_{g_j\in A_2} v_{2jk}(1 - 2x^*_j)\\
&\geq \textstyle\sum_{g_j\in A_1} v_{2jk}(2x^*_j - 1) \geq \sum_{g_j\in I_1} v_{2jk} = v_2(A_1 \setminus F_1)_k.
\end{aligned}$$

Thus, any envy can be eliminated by removing all the fractional items from the bundle of the other agent. Hence, the allocation is strong sEF$(2\ell-1)$. Since $c \geq 2\ell - 1$, the allocation is strong sEFc as well.

Finally, since the LP has polynomial number of constraints and variables, a basic optimal solution can be computed in polynomial time via standard linear programming techniques (see, e.g., a textbook [8]). Hence, the resulting allocation $\mathbf{A}$ can be found in polynomial time. ◀

If the valuations of the two agents are identical, we can reduce the requirement of $2\ell - 1$ to ℓ. Due to space constraints, the proof is deferred to the Appendix.

▶ **Theorem 8.** *When there are two identical agents and c is at least the number of dimensions (i.e., $c \geq \ell$), a strong sEFc allocation always exists. Moreover, such an allocation can be computed in polynomial time.*

When the number of agents is three or more, a direct application of the polytope-based approach presented in Theorem 7 and Theorem 8 fails. Indeed, consider the case with three agents in one dimension, where the valuations are given by

$$(v_{i11}, v_{i21}, v_{i31}, \ldots, v_{im1}) = (2(m-1), 1, 1, \ldots, 1) \qquad (i = 1, 2, 3).$$

In this case, allocating half of item g_1 to agents 1 and 2, while assigning all the remaining items to agent 3 yields an envy-free fractional allocation. A reasonable rounding method in this situation is to assign item g_1 to either agent 1 or 2. However, the agent who does not receive g_1 will envy agent 3, and this envy cannot be eliminated by removing $m - 2$ items.

To overcome this difficulty, we pre-assign certain items before employing the polytope-based approach. This strategy allows us to derive an upper bound on c that guarantees the existence of a strong sEFc allocation independent of the total number of items m.

▶ **Theorem 9.** *When c is at least the square of the product of the numbers of agents and the number of dimensions (i.e., $c \geq n^2\ell^2$), a strong sEFc allocation always exists. Moreover, such an allocation can be computed in polynomial time.*

Proof. Let $(N, M, L, (v_i)_{i \in N})$ be a given instance of the MDFA problem. We assume that $N = [n]$, $M = \{g_1, \ldots, g_m\}$, and $L = [\ell]$. If $m \leq n \cdot c$, then simply allocating at most c items to each agent yields a strong EFc allocation. Hence, in what follows, we assume that $m > n \cdot c$ $(\geq n^3\ell^2)$.

For each agent $i \in N$ and dimension $k \in L$, we begin by sequentially allocating the $(n-1)^2\ell$ most desired items in dimension k to agent i. Denote by $Z_{i,k}$ the set of items allocated to agent i with respect to dimension k. This procedure is described in Algorithm 1. Intuitively, the set $Z_{i,k}$ is used to compensate for the loss of value in the kth dimension that occurs when rounding a fractional solution. Let $R \coloneqq M \setminus \bigcup_{i \in N} \bigcup_{k \in L} Z_{i,k}$ be the set of remaining items. Note that, for every $g_j \in Z_{i,k}$ and $g_{j'} \in R$, we have $v_{ijk} \geq v_{ij'k}$.

Algorithm 1 Pre-assignment of items.

```
Let R ← M;
foreach i ∈ N and k ∈ L do
    Initialize Z_{i,k} ← ∅;
    for p ← 1 to (n − 1)^2 ℓ do
        Choose g ∈ arg max_{g_j ∈ R} v_{ijk};
        Update Z_{i,k} ← Z_{i,k} ∪ {g} and R ← R \ {g};
```

Consider the following polytope P in $\mathbb{R}^{N\times R}$:

$$P := \left\{ x \in \mathbb{R}^{N\times R} : \begin{array}{ll} \sum_{g\in R} v_i(g)_k(x_{ij} - x_{i'j}) \geq 0 & (\forall i, i' \in N \text{ with } i \neq i',\ \forall k \in L), \\ \sum_{i\in N} x_{ig} = 1 & (\forall g \in R), \\ x_{ig} \geq 0 & (\forall i \in N,\ \forall g \in R) \end{array} \right\}.$$

A feasible point $x \in P$ is a simultaneously proportional fractional allocation of R, where x_{ig} is interpreted as the fraction of item g allocated to agent i. Note that, since the uniform allocation $x = (1/n)_{i\in N, g\in R}$ is feasible, the polytope P is nonempty.

Let $m' = |R|$. Since P is in $m'n$-dimensional space, any extreme point is determined by $m'n$ linearly independent tight constraints. In particular, at least $m'n - m' - n(n-1)\ell$ of these tight constraints must be zero. This implies that every extreme point has at most $m' + n(n-1)\ell$ nonzero coordinates. Consequently, at least $m' - n(n-1)\ell$ items are allocated integrally (i.e., for such an item $g \in R$, there exists some agent i with $x_{ig} = 1$).

Now, let x^* be an extreme point of P, and for each agent $i \in N$, define $I_i := \{g \in R : x^*_{ig} = 1\}$. Let $F := R \setminus \bigcup_{i\in N} I_i$ be the set of items that are not allocated integrally. By the above discussion, we have $|F| \leq n(n-1)\ell$. Partition F arbitrary into n subsets $(F_1, \ldots, F_n)$ such that $|F_i| \leq (n-1)\ell$ for all $i \in N$. Then, define the allocation $\mathbf{A} = (A_1, \ldots, A_n)$ by $A_i = \left(\bigcup_{k\in L} Z_{i,k}\right) \cup I_i \cup F_i$ for each $i \in N$. We now show that $\mathbf{A}$ is a strong sEFc allocation if $c \geq n^2\ell^2$.

For every $i, i' \in N$ and every $k \in L$, we have

$$\begin{aligned} v_i(A_i)_k &\geq v_i(I_i)_k + v_i(F_i)_k + v_i(Z_{i,k})_k \geq v_i(I_{i'})_k - v_i(F)_k + v_i(F_i)_k + v_i(Z_{i,k})_k \\ &= v_i\left(A_{i'} \setminus \left(F_{i'} \cup \bigcup_{k'\in L} Z_{i',k'}\right)\right)_k - v_i(F \setminus F_i)_k + v_i(Z_{i,k})_k \\ &\geq v_i\left(A_{i'} \setminus \left(F_{i'} \cup \bigcup_{k'\in L} Z_{i',k'}\right)\right)_k, \end{aligned}$$

where the last inequality follows from the facts that, for every $g_j \in Z_{i,k}$ and $g_{j'} \in F \subseteq R$, $v_{ijk} \geq v_{ij'k}$, and $|F \setminus F_i| \leq (n-1)^2\ell \leq |Z_{i,k}|$. Therefore, we obtain that $\mathbf{A}$ is a strong sEFc when $c = n^2\ell^2$ since

$$n^2\ell^2 \geq (n-1)\ell + \ell \cdot (n-1)^2\ell \geq |F_{i'}| + \textstyle\sum_{k'\in L} |Z_{i',k'}| = \left|F_{i'} \cup \bigcup_{k'\in L} Z_{i',k'}\right|$$

for every $i' \in N$.

Finally, since the polytope P has polynomial number of constraints and variables, a basic optimal solution can be computed in polynomial time via standard linear programming techniques. Hence, the resulting allocation $\mathbf{A}$ can be found in polynomial time. ◀

Next, we provide a dynamic programming algorithm that checks existence of a weak sEFc allocation. The key idea is to process the items one by one, gradually building up partial allocations and summarizing their "state" in a way that keeps track of all the information necessary to verify the weak sEFc condition later.

▶ **Theorem 10.** *The existence of a weak sEFc allocation can be checked in time $m^{O(n^2\ell)} \cdot (v_{\max}+1)^{O(cn^2\ell)}$, where $v_{\max} := \max_{i\in N} \max_{g_j\in M} \max_{k\in L} v_{ijk}$. In particular, if c, n, and ℓ are constant and the valuations are encoded in unary, the existence of a weak sEFc can be checked in polynomial time. Similarly, if ℓ and n are constant, $c = O(\log m)$, and the valuations are binary, then the existence of a strong sEFc can be checked in polynomial time.*

Proof. Let $(N, M, L, (v_i)_{i\in N})$ be an instance of the MDFA problem. We assume that $N = [n]$, $M = \{g_1, \ldots, g_m\}$, $L = [\ell]$, and that for each agent $i \in N$ and item $g \in M$. For each $j \in \{0, 1, \ldots, m\}$, let $M_j = \{g_1, g_2, \ldots, g_j\}$ denote the prefix of items. Without loss of generality, we assume that $m > n \cdot c$ since otherwise simply allocating at most c items to each agent yields a weak EFc allocation.

For each $j \in \{0, 1, \ldots, m\}$ and for each allocation $\mathbf{A}$ of M_j, we summarize its state (V, T) as follows:

- $V \in \mathbb{Z}^{N \times N \times L}$ represents the valuation profile. For every $i, i' \in N$ and every $k \in L$, $V_{i,i',k} = v_i(A_{i'})_k \in [0, 1, \ldots, m \cdot v_{\max}]$.
- $T \in \mathbb{Z}^{N \times N \times L \times [c]}$ encodes the top c item values. For every $i, i' \in N$, every $k \in L$, and $s \in [c]$, let $T_{i,i',k,s}$ denote the sth largest value in the multiset $\{v_i(g) : g \in A_{i'}\}$. If $|A_{i'}| < c$, the remaining entries in $T_{i,i',k}$ are filled with 0.

We now describe a dynamic programming algorithm that computes a table $\mathrm{DP}[j][V][T]$, which is a Boolean indicator that is set to `true` if there exists an allocation of the items in M_j that achieves the state (V, T), and `false` otherwise. Notice that the size of the DP table is at most

$$(m+1) \cdot (m \cdot v_{\max} + 1)^{n \cdot n \cdot \ell} \cdot (v_{\max} + 1)^{c \cdot n \cdot n \cdot \ell} = m^{O(n^2 \ell)} \cdot (v_{\max} + 1)^{O(cn^2 \ell)}.$$

Base case. For $j = 0$ (i.e., no items are allocated), the table $\mathrm{DP}[0][V][T]$ is `true` only when the valuations V and the top c item values T are all zero. In other words, the DP table can be filled as follows:

$$\mathrm{DP}[0][V][T] = \begin{cases} \texttt{true} & \text{if } V_{i,i'k} = T_{i,i',k,s} = 0 \text{ for all } i, i' \in N,\ k \in L, \text{ and } s \in [c], \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Inductive Step. For $j \geq 1$, assume that $\mathrm{DP}[j-1][V][T] = \texttt{true}$ for some state (V, T) corresponding to an allocation of M_{j-1}. We then process item g_j. For each agent $i \in N$, consider allocating g_j to i. For each such choice, we update the state as follows:

- **Update V:** For every $i', i'' \in N$ and every $k \in L$, define

$$V'_{i',i'',k} = \begin{cases} V_{i',i'',k} + v_{i'jk} & \text{if } i'' = i, \\ V_{i',i'',k} & \text{otherwise.} \end{cases}$$

  This corresponds to the valuation profile of the allocation where item g_j is additionally assigned to agent i.
- **Update T:** For every $i' \in N$ and every $k \in L$, define $T'_{i',i,k}$ by updating $T_{i',i,k}$ by inserting the value $v_{i'jk}$ into the multiset $\{T_{i',i,k,1}, T_{i',i,k,2}, \ldots, T_{i',i,k,c}\}$ and retaining only the c highest entries. For every agent $i'' \neq i$, $T'_{i',i'',k}$ is defined to be $T_{i',i'',k}$.

Note that this update can be performed in $O(n^2 \cdot \ell \cdot c)$ time.

After performing these updates for every state (V, T) with $\mathrm{DP}[j-1][V][T] = \texttt{true}$ and for every choice of agent $i \in N$ who receives g_j, we set the corresponding entry $\mathrm{DP}[j][V'][T']$ to `true`. All other cells are set to `false`.

After processing all items, the table $\mathrm{DP}[m][\cdot][\cdot]$ represents possible states corresponding to full allocations of M. We then examine each state (V, T) such that $\mathrm{DP}[m][V][T] = \texttt{true}$ to verify the weak sEFc condition. Specifically, for every pair of distinct agents $i, i' \in N$ and for every dimension $k \in L$, weak sEFc requires that there exists a set $X \subseteq A_{i'}$, with $|X| \leq c$, such that $v_i(A_i)_k \geq v_i(A_{i'})_k - \sum_{g \in X} v_i(g)_k$. Since the list $T_{i,i',k}$ contains the c highest values among the multiset $\{v_i(g)_k : g \in A_{i'}\}$, it suffices to verify that $V_{iik} \geq V_{ii'k} - \sum_{s=1}^{c} T_{i,i',k,s}$ for every distinct $i, i' \in N$ and every $k \in L$. This condition can be checked in $O(n^2 \cdot \ell \cdot c)$ time. A weak sEFc allocation exists if and only if at least one state (V, T) such that $\mathrm{DP}[m][V][T] = \texttt{true}$ satisfies this inequality. Finally, the overall running time is

$$m^{O(n^2 \ell)} \cdot (v_{\max} + 1)^{O(cn^2 \ell)} \cdot n \cdot O(n^2 \cdot \ell \cdot c) = m^{O(n^2 \ell)} \cdot (v_{\max} + 1)^{O(cn^2 \ell)},$$

since computing the DP table dominates the running time. This completes the proof. ◀

Finally, we provide a dynamic programming algorithm that checks the existence of a strong sEFc allocation. Unlike the approach for weak sEFc, we first enumerate all possible combinations of subsets used to eliminate envies.

▶ **Theorem 11.** *The existence of a strong sEFc allocation can be checked in time $m^{O(n^2(c+\ell))} \cdot (v_{\max}+1)^{O(n^2\ell)}$, where $v_{\max} \coloneqq \max_{i\in N} \max_{g_j\in M} \max_{k\in L} v_{ijk}$. In particular, if c, n, and ℓ are constant and the valuations are encoded in unary, then the existence of a strong sEFc can be checked in polynomial time.*

Proof. Let $(N, M, L, (v_i)_{i\in N})$ be an instance of the MDFA problem. We assume that $N = [n]$, $M = \{g_1, \dots, g_m\}$, $L = [\ell]$, and that for each agent $i \in N$ and item $g \in M$. Without loss of generality, we assume that $m > n \cdot c$ since otherwise simply allocating at most c items to each agent yields a strong EFc allocation.

We enumerate a collection $(X_{ii'})_{i,i'\in N}$ with $|X_{ii'}| \le c$ for all $i, i' \in N$. Each $X_{ii'}$ represents the subset that is used to eliminate envy from agent i towards agent i'. We consider allocations $\mathbf{A}$ that satisfies $A_i \supseteq \bigcup_{i'\in N} X_{ii'}$ for all agent $i \in N$. To satisfy consistency, for distinct agents i and ι, the sets of items used for envy elimination must be disjoint, i.e., $\left(\bigcup_{i'\in N} X_{ii'}\right) \cap \left(\bigcup_{\iota'\in N} X_{\iota\iota'}\right) = \emptyset$. Note that the number of possible choices for the collection $(X_{ii'})_{i,i'\in N}$ is at most $\prod_{i,i'\in N}(1+m)^c = m^{O(cn^2)}$.

Fix some collection $(X_{ii'})_{i,i'\in N}$ satisfying $|X_{ii'}| \le c$ for all $i, i' \in N$ and $X_{ii'} \cap X_{\iota\iota'} = \emptyset$ for all i, i', ι, ι' with $i \neq \iota$. Define $X_i = \bigcup_{i'\in N} X_{ii'}$ for each $i \in N$ and $X = \bigcup_{i\in N} X_i$. Let $M' = M \setminus X$ be the set of remaining items. Write M' as $\{g_{\sigma(1)}, \dots, g_{\sigma(m')}\}$ and for each $j \in \{0, 1, \dots, m'\}$, let $M'_j = \{g_{\sigma(1)}, g_{\sigma(2)}, \dots, g_{\sigma(j)}\}$ denote the prefix of items.

For each $j \in \{0, 1, \dots, m'\}$ and for each allocation $\mathbf{A}$ of $X \cup M'_j$ with $A_i \supseteq \bigcup_{i'\in N} X_{ii'}$ for all $i \in N$, we keep the valuation profile $V = (v_i(A_{i'})_k)_{i,i'\in N,\, k\in L}$.

We now describe a dynamic programming algorithm that computes a table DP$[j][V]$, which is a Boolean indicator that is set to `true` if there exists an allocation of the items in $X \cup M'_j$ that achieves the valuation profile V, and `false` otherwise. Notice that the size of the DP table is at most $(m+1) \cdot (m \cdot v_{\max}+1)^{n\cdot n\cdot \ell} = m^{O(n^2\ell)} \cdot (v_{\max}+1)^{O(n^2\ell)}$.

Base case. For $j = 0$ (i.e., no items are allocated), the table DP$[0][V]$ is `true` only when the valuation profile V correspond to the allocation $(X_1, \dots, X_n)$. In other words, the DP table can be filled as follows:

$$\mathrm{DP}[0][V] = \begin{cases} \texttt{true} & \text{if } V_{i,i'k} = \sum_{g_j\in X_{i'}} v_{ijk} \text{ for all } i, i' \in N,\ k \in L, \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Inductive Step. For $j \ge 1$, assume that DP$[j-1][V] =$ `true` for some valuation profile V corresponding to an allocation of $X \cup M'_{j-1}$. We then process item $g_{\sigma(j)}$. For each agent $i \in N$, consider allocating $g_{\sigma(j)}$ to i. For each such choice, we construct an updated valuation profile V' as follows:

$$V'_{i',i'',k} = \begin{cases} V_{i',i'',k} + v_{i'jk} & \text{if } i'' = i, \\ V_{i',i'',k} & \text{otherwise,} \end{cases} \quad \text{for every } i', i'' \in N \text{ and every } k \in L.$$

This corresponds to the valuation profile of the allocation where item $g_{\sigma(j)}$ is additionally assigned to agent i. After performing these updates for every valuation profile V with DP$[j-1][V] =$ `true` and for every choice of agent $i \in N$ who receives $g_{\sigma(j)}$, we set the corresponding entry DP$[j][V']$ to `true`. All other cells are set to `false`.

After processing all items, the table DP$[m][\cdot]$ represents possible valuation profile corresponding to full allocations of M. We then examine each valuation profile V such that DP$[m][V]$ = `true` to verify the strong sEFc condition. Specifically, for every pair of distinct agents $i, i' \in N$ and for every dimension $k \in L$, it suffices to verify that $V_{iik} \geq V_{ii'k} - \sum_{j \in g_j \in X_{i,i'}} v_{ijk}$.

This condition can be checked in $O(n^2 \cdot \ell \cdot c)$ time. A strong sEFc allocation $\mathbf{A}$ corresponding to $(X_{ii'})_{i,i' \in N}$ exists if and only if at least one state V such that DP$[m][V]$ = `true` satisfies this inequality.

Finally, the overall running time is

$$m^{O(cn^2)} \cdot m^{O(n^2\ell)} \cdot (v_{\max}+1)^{O(n^2\ell)} \cdot n \cdot O(n^2 \cdot \ell \cdot c) = m^{O(n^2(c+\ell))} \cdot (v_{\max}+1)^{O(n^2\ell)}.$$

This completes the proof. ◀

5 Hardness of Checking Existence

In this section, we show that checking the existence of a weak and a strong sEF1 allocation are NP-hard. Moreover, we demonstrate the NP-hardness of checking whether a given allocation is strongly sEFc.

We first prove that checking whether a weak sEF1 allocation exists is strongly NP-complete, even in the case of identical binary valuations with two agents.

▶ **Theorem 12.** *Checking the existence of a weak sEF1 allocation is strongly NP-complete even for identical binary valuations with two agents.*

Proof. The problem is in NP, since we can verify the condition of weak sEF1 for a given allocation in polynomial time.

We prove strong NP-hardness by reduction from the monotone not-all-equal 3-SAT (MNAE3SAT) problem, which is known to be strongly NP-complete [31]. In the problem, we are given n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. Each clause C_k is of the form $C_k = (z_{k,1}, z_{k,2}, z_{k,3})$, where every literal $z_{k,1}, z_{k,2}, z_{k,3}$ is one of the variables $x_1, x_2, \ldots, x_n$. The goal is to decide whether there exists a truth assignment to the variables such that, in each clause, the literals do not all share the same truth value; in other words, each clause must contain at least one true literal and at least one false literal.

From a given instance of the MNAE3SAT problem, we construct an instance of the MDFA problem as follows. Let $N = \{1, 2\}$, $M = \{g_1, g_2, \ldots, g_n\}$, and $L = \{1, 2, \ldots, m\}$. For every agent $i \in N$, item $g_j \in M$, and dimension $k \in L$, define

$$v_{ijk} = \begin{cases} 1 & \text{if } x_j \in \{z_{k,1}, z_{k,2}, z_{k,3}\}, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that in every dimension k (which corresponds to clause C_k), exactly the three items corresponding to the three variables appearing in C_k are assigned value 1, and all other items are assigned 0. Moreover, both agents have the same binary valuation functions.

We now show that the constructed instance has a weak sEF1 alocation if and only if the MNAE3SAT instance is a yes-instance.

($\Rightarrow$) Suppose that a weak sEF1 allocation $\mathbf{A} = (A_1, A_2)$ exists for the reduced instance. In the constructed instance, for each dimension $k \in L$, exactly three items (corresponding to the three variables of clause C_k) have value 1. Observe that if, for some dimension k, all these three items are allocated to the same agent, then one agent would have a value of 0

while the other has a value of 3 in that dimension. Removing one item (value 1) from the bundle with three would yield a value of 2, which would still leave an envy. Thus, to satisfy the weak sEF1 condition, it must be that in every dimension k both agents receive at least one item with value 1.

Now define a truth assignment for the MNAE3SAT instance by assigning x_j to be `true` if $g_j \in A_1$ and `false` otherwise. Since for every clause C_k the corresponding three items (with value 1 in dimension k) are split between the two agents, the clause contains at least one literal that is true and at least one that is false. Hence, the constructed truth assignment satisfies every clause in the MNAE3SAT instance.

($\Leftarrow$) Conversely, suppose that the given MNAE3SAT instance is a yes-instance. Let $I \subseteq [n]$ be the set of indices of variables assigned *true* in some satisfying truth assignment, so that in each clause at least one variable is true and at least one is false. We construct an allocation $\mathbf{A} = (A_1, A_2)$ by letting $A_1 = \{g_j \in M : j \in I\}$ and $A_2 = \{g_j \in M : j \notin I\}$. Consider any dimension $k \in L$ (i.e., any clause C_k). Since $C_k = (z_{k,1}, z_{k,2}, z_{k,3})$, the three items with value 1 in dimension k are precisely those items corresponding to the variables in C_k. Because the truth assignment is satisfying in the MNAE3SAT sense, at least one of these variables is true and at least one is false. Hence, in dimension k the items with value 1 are split between A_1 and A_2. In particular, one bundle will contain one item valued 1 on dimension k while the other will contain two.

Now, note that under our binary valuation, if an agent's bundle in dimension k yields a total value of 1 while the other agent's bundle yields 2, then there is envy. However, by removing a single item (with value 1 in dimension k) from the envied bundle, the value drops from 2 to 1, thus eliminating the envy in that dimension. Since this argument applies to every dimension (clause), the allocation $\mathbf{A}$ is weak sEF1.

Since the reduction can be performed in polynomial time and the existence of a weak sEF1 allocation is equivalent to the satisfiability of the given MNAE3SAT instance, checking the existence of a weak sEF1 allocation is NP-hard even when there are two agents. ◀

Next, we prove that verifying the existence of a strong sEF1 allocation is NP-complete, even for the identical binary case with two agents and two dimensions.

▶ **Theorem 13.** *Checking the existence of a strong sEF1 allocation is weakly NP-complete even when there are only two identical agents and the number of dimensions is two.*

Proof. The problem is in NP, since we can verify the condition of strong sEF1 for a given allocation in polynomial time.

We prove NP-hardness by reduction from the PARTITION problem, which is known to be NP-complete [16]. In the Partition problem, we are given n positive integers $a_1, a_2, \ldots, a_n$ and an integer T such that $\sum_{j=1}^{n} a_j = 2T$. The goal is to determine whether there exists a partition (S_1, S_2) of the set $[n]$ such that $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = T$.

Given an instance of the PARTITION problem, we construct an instance of the MDFA problem as follows. Let $N = \{1, 2\}$, $M = \{g_1, g_2, \ldots, g_{n+2}\}$, and $L = \{1, 2\}$. Both agent have identical valuation functions. For each item g_j with $j \in [n]$, the value is a_j in both dimensions. Item g_{n+1} has a very high value $2T + 1$ in the first dimension and zero in the second, while item g_{n+2} has a value of $2T + 1$ for the second dimension and zero in the first. The valuations are summarized in Table 3.

We now show that the constructed instance has a strong sEF1 allocation if and only if the PARTITION instance is a yes-instance.

Table 3 The valuations for the reduced instance in Theorem 13.

	g_1	g_2	$\cdots$	g_n	g_{n+1}	g_{n+2}
$v_i(g_j)_1$	a_1	a_2	$\cdots$	a_n	$2T+1$	0
$v_i(g_j)_2$	a_1	a_2	$\cdots$	a_n	0	$2T+1$

(⇒) Suppose that there exists a strong sEF1 allocation $\mathbf{A} = (A_1, A_2)$ for the constructed instance. Note that if both extra items g_{n+1} and g_{n+2} were assigned to the same agent, then to eliminate envy the other agent would need to have both items removed, which violates the strong sEF1 condition. Thus, g_{n+1} and g_{n+2} must be assigned to different agents. Without loss of generality, assume that agent 1 receives g_{n+1} and agent 2 receives g_{n+2}. Define

$$R_1 = \textstyle\sum_{g_j \in A_1 \setminus \{g_{n+1}\}} a_j \qquad \text{and} \qquad R_2 = \sum_{g_j \in A_2 \setminus \{g_{n+2}\}} a_j.$$

Then, the total values for each agent in each dimension are

$$v_1(A_1)_1 = v_2(A_1)_1 = 2T + 1 + R_1, \qquad v_1(A_1)_2 = v_2(A_1)_2 = R_1,$$
$$v_1(A_2)_1 = v_2(A_2)_1 = R_2, \qquad v_1(A_2)_2 = v_2(A_2)_2 = 2T + 1 + R_2.$$

To eliminate envy from agent 1 to agent 2 in the second dimension, it holds that $v_1(A_1)_2 \geq \min_{g \in A_2} v_1(A_2 \setminus \{g\})_2$. Thus, we have

$$R_1 = v_1(A_1)_2 \geq \min_{g \in A_2} v_1(A_2 \setminus \{g\})_2 = v_1(A_2 \setminus \{g_{n+2}\})_2 = R_2.$$

Similarly, to eliminate envy from agent 2 to agent 1 in the first dimension, we have

$$R_2 = v_2(A_2)_1 \geq \min_{g \in A_1} v_2(A_1 \setminus \{g\})_1 = v_2(A_1 \setminus \{g_{n+1}\})_1 = R_1.$$

Together these yield $R_1 = R_2 = T$. This means that the corresponding partition $(\{j \in [n] : g_j \in A_1\},\ \{j \in [n] : g_j \in A_2\})$ is a solution of the PARTITION instance.

(⇐) Conversely, suppose that the given PARTITION instance is a yes-instance. Then, there exists a partition (S_1, S_2) of $[n]$ such that $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = T$. Define an allocation $\mathbf{A} = (A_1, A_2)$ by $A_1 = \{g_j : j \in S_1\} \cup \{g_{n+1}\}$ and $A_2 = \{g_j : j \in S_1\} \cup \{g_{n+2}\}$. Under this allocation, the total value for each agent in each dimension is:

$$v_1(A_1)_1 = v_2(A_1)_1 = 3T + 1, \qquad v_1(A_1)_2 = v_2(A_1)_2 = T,$$
$$v_1(A_2)_1 = v_2(A_2)_1 = T, \qquad v_1(A_2)_2 = v_2(A_2)_2 = 3T + 1.$$

Let us verify the strong sEF1 conditions:

$$v_1(A_1)_1 \geq v_1(A_2)_1, \qquad v_1(A_1)_2 \geq v_1(A_2 \setminus \{g_{n+2}\})_2,$$
$$v_2(A_2)_1 = v_2(A_1 \setminus \{g_{n+1}\})_1, \qquad v_2(A_2)_2 \geq v_2(A_1)_2.$$

Thus, removing a single item suffices to eliminate envy in every case, so the allocation is strong sEF1.

Since the reduction can be performed in polynomial time and the existence of a strong sEF1 allocation is equivalent to the answer of the given PARTITION instance, checking the existence of a strong sEF1 allocation is NP-hard even when there are two identical agents and two dimensions. ◀

▶ Remark 14. A reduction analogous to that used in the proof of Theorem 13 shows that if checking the existence of an EF allocation is NP-hard in a certain one-dimensional setting, then the corresponding problem of checking the existence of a strong EFc allocation in nc dimensions is also NP-hard. Specifically, given an instance $(N = [n], M = \{g_1, \dots, g_m\}, (v_i)_{i \in N})$ of the one-dimensional EF allocation existence problem, we construct an instance of the MDFA problem $(\hat{N}, \hat{M}, \hat{L}, (\hat{v}_i)_{i \in N})$ as follows: set $\hat{N} = N$, $\hat{M} = M \cup \{g_{m+1}, \dots, g_{m+nc}\}$, $\hat{L} = [nc]$, and define

$$\hat{v}_{ijk} = \begin{cases} v_{ij} & \text{if } j \leq m, \\ \sum_{j'=1}^{m} v_{ij'} + 1 & \text{if } j > m \text{ and } j - m = k, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for all } i \in \hat{N}, g_j \in \hat{M}, k \in \hat{L}.$$

Using this reduction, we can, for example, prove that checking the existence of a strong sEFc allocation is strongly NP-hard even when the valuations are identical (since it is not difficult to deduce strong NP-hardness for the one-dimensional EF allocation existence problem of identical case by a reduction from the 3-PARTITION problem).

Finally, we show that determining whether a given allocation is strong sEFc is NP-complete when c is a parameter. Due to space constraints, the proof is deferred to the Appendix.

▶ **Theorem 15.** *Checking whether a given allocation is strong sEFc is strongly NP-complete, even for the identical binary case with two agents, when c is part of the input.*

6 Conclusion

In this paper, we introduced the problem of allocating indivisible items in a multidimensional setting, where each agent's valuation is represented by an ℓ-dimensional vector. We explored two notions of relaxed envy-freeness tailored to this setting, which we call weak sEFc and strong sEFc. We analyzed upper and lower bounds on c that guarantee the existence of a weak sEFc or a strong sEFc allocation. In addition, we developed algorithms to determine whether a weak sEFc or a strong sEFc allocation exists, and we established NP-hardness results for checking the existence of weak sEF1 and strong sEF1 allocations. Moreover, we demonstrated that determining whether a given allocation is strong sEFc is NP-complete when c is a parameter.

An important direction for future work is to narrow the gap between the upper and lower bounds on c required to guarantee the existence of a weak sEFc or strong sEFc allocation. In particular, it remains an open question whether a weak sEF1 allocation always exists when the number of dimensions is two and the number of agents is at least 3. Another promising direction is to explore the existence and computational complexity of other fairness and efficiency notions in multidimensional settings, such as proportionality. For example, we can define multidimensional version of (relaxed) proportionalities as follows:

- An allocation $\mathbf{A}$ is said to be *weak simultaneously proportional up to c goods (weak sPROPc)* if for any agent $i \in N$ and any dimension $k \in L$, there exists a set $X \subseteq M \setminus A_i$ such that $|X| \leq c$ and $v_i(A_i \cup X)_k \geq v_i(M)_k/n$.
- An allocation $\mathbf{A}$ is said to be *strong simultaneously proportional up to c goods (strong sPROPc)* if for any agents $i \in N$, there exists a set $X \subseteq M \setminus A_i$ such that $|X| \leq c$ and $v_i(A_i \cup X)_k \geq v_i(M)_k/n$ for any dimension $k \in L$.

Note that weak sPROPc was introduced by Bu et al. [10], but strong sPROPc has not been studied. For these notions of proportionality, the polytope-based approach would yield a meaningful guarantee; however, whether stronger guarantees can be achieved remains an interesting avenue for future work.

Finally, we believe that a similar analysis should be possible for the chore allocation case, but we leave this for future research.

References

1 Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

2 Georgios Amanatidis, Haris Aziz, Georgios Birmpas, Aris Filos-Ratsikas, Bo Li, Hervé Moulin, Alexandros A. Voudouris, and Xiaowei Wu. Fair division of indivisible goods: Recent progress and open questions. *Artif. Intell.*, 2022.

3 Haris Aziz. Computational social choice: Some current and new directions. In *IJCAI*, pages 4054–4057, 2016. URL: http://www.ijcai.org/Abstract/16/599.

4 Haris Aziz, Bo Li, Hervé Moulin, and Xiaowei Wu. Algorithmic fair allocation of indivisible items: A survey and new questions. *ACM SIGecom Exchanges*, 20(1):24–40, 2022. doi:10.1145/3572885.3572887.

5 Haris Aziz and Simon Rey. Almost group envy-free allocation of indivisible goods and chores. *ArXiv*, abs/1907.09279, 2019. arXiv:1907.09279.

6 Siddharth Barman, Soroush Ebadian, Mohamad Latifian, and Nisarg Shah. Fair division with market values. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 13589–13596, 2025. doi:10.1609/AAAI.V39I13.33484.

7 Marcus Berliant, William Thomson, and Karl Dunz. On the fair division of a heterogeneous commodity. *Journal of Mathematical Economics*, 21(3):201–216, 1992.

8 Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

9 Steven J Brams and Alan D Taylor. An envy-free cake division protocol. *The American Mathematical Monthly*, 102(1):9–18, 1995.

10 Xiaolin Bu, Zihao Li, Shengxin Liu, Jiaxin Song, and Biaoshuai Tao. Fair division with allocator's preference. In *Web and Internet Economics*, pages 77–94. Springer Nature Switzerland, 2024.

11 Ioannis Caragiannis, Kasper Green Larsen, and Sudarshan Shyam. A new lower bound for multi-color discrepancy with applications to fair division. *arXiv preprint arXiv:2502.10516*, 2025. doi:10.48550/arXiv.2502.10516.

12 John Cloutier, Kathryn Nyman, and Francis Edward Su. Two-player envy-free multi-cake division. *Math. Soc. Sci.*, 59:26–37, 2009. doi:10.1016/J.MATHSOCSCI.2009.09.002.

13 Vincent Conitzer, Rupert Freeman, Nisarg Shah, and Jennifer Wortman Vaughan. Group fairness for the allocation of indivisible goods. In *AAAI Conference on Artificial Intelligence*, 2019.

14 Michele Flammini, Gianluigi Greco, and Giovanna Varricchio. Fair division with social impact. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 13856–13863, 2025. doi:10.1609/AAAI.V39I13.33515.

15 Duncan Karl Foley. *Resource allocation and the public sector*. Yale Economics Essays, 1966.

16 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

17 Paul W Goldberg, Alexandros Hollender, Ayumi Igarashi, Pasin Manurangsi, and Warut Suksompong. Consensus halving for sets of items. *Mathematics of Operations Research*, 47(4):3357–3379, 2022. doi:10.1287/MOOR.2021.1249.

18 Hadi Hosseini, Ayumi Igarashi, and Andrew Searns. Fair division of time: Multi-layered cake cutting. In *International Joint Conference on Artificial Intelligence*, 2020.

19 Marco D Huesch. One and done? equality of opportunity and repeated access to scarce, indivisible medical resources. *BMC Medical Ethics*, 13:1–13, 2012.

20 Ayumi Igarashi and Frédéric Meunier. Envy-free division of multi-layered cakes. In *International Conference on Web and Internet Economics*, pages 504–521. Springer, 2021. doi:10.1007/978-3-030-94676-0_28.

21 Yasushi Kawase, Bodhayan Roy, and Mohammad Azharuddin Sanpui. Resource allocation under the latin square constraint. *arXiv preprint arXiv:2501.06506*, 2025. doi:10.48550/arXiv.2501.06506.

22 Maria Kyropoulou, Warut Suksompong, and Alexandros A Voudouris. Almost envy-freeness in group resource allocation. *Theoretical Computer Science*, 841:110–123, 2020. doi:10.1016/J.TCS.2020.07.008.

23 Nicolas Lebert, Frédéric Meunier, and Quentin Carbonneaux. Envy-free two-player mm-cake and three-player two-cake divisions. *Oper. Res. Lett.*, 41:607–610, 2013. doi:10.1016/J.ORL.2013.07.010.

24 Richard J Lipton, Evangelos Markakis, Elchanan Mossel, and Amin Saberi. On approximately fair allocations of indivisible goods. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 125–131, 2004. doi:10.1145/988772.988792.

25 Pasin Manurangsi and Raghu Meka. Tight lower bound for multicolor discrepancy. *arXiv preprint arXiv:2504.18489*, 2025. doi:10.48550/arXiv.2504.18489.

26 Pasin Manurangsi and Warut Suksompong. Asymptotic existence of fair divisions for groups. *Mathematical Social Sciences*, 89:100–108, 2017. doi:10.1016/J.MATHSOCSCI.2017.05.006.

27 Pasin Manurangsi and Warut Suksompong. Almost envy-freeness for groups: Improved bounds via discrepancy theory. *Theor. Comput. Sci.*, 930:179–195, 2021. doi:10.1016/J.TCS.2022.07.022.

28 Eric S Maskin. On the fair allocation of indivisible goods. In *Arrow and the Foundations of the Theory of Economic Policy*, pages 341–349. Springer, 1987.

29 Kathryn Nyman, Francis Edward Su, and Shira Zerbib. Fair division with multiple pieces. *Discret. Appl. Math.*, 283:115–122, 2017.

30 Stanley Reiter and Gordon R Sherman. Allocating indivisible resources affording external economies or diseconomies. *International Economic Review*, 3(1):108–135, 1962.

31 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978. doi:10.1145/800133.804350.

32 Erel Segal-Halevi and Shmuel Nitzan. Fair cake-cutting among families. *Social Choice and Welfare*, 53:709–740, 2015.

33 Erel Segal-Halevi and Warut Suksompong. Democratic fair allocation of indivisible goods. *Artificial Intelligence*, 277:103167, 2019. doi:10.1016/J.ARTINT.2019.103167.

34 Warut Suksompong. Approximate maximin shares for groups of agents. *Math. Soc. Sci.*, 92:40–47, 2017. doi:10.1016/J.MATHSOCSCI.2017.09.004.

35 Warut Suksompong. On the number of almost envy-free allocations. *Discrete Applied Mathematics*, 284:606–610, 2020. doi:10.1016/J.DAM.2020.03.039.

A Omitted proofs

▶ **Theorem 8.** *When there are two identical agents and c is at least the number of dimensions (i.e., $c \geq \ell$), a strong sEFc allocation always exists. Moreover, such an allocation can be computed in polynomial time.*

Proof. Let $(N, M, L, (v_i)_{i \in N})$ be a given instance of the MDFA problem. We assume that $N = \{1, 2\}$, $M = \{g_1, \ldots, g_m\}$, $L = [\ell]$, and $v_1 = v_2$. Consider the following polytope in $\mathbb{R}^m$:

$$P := \left\{ x \in \mathbb{R}^m : \begin{array}{ll} \sum_{j=1}^{m} v_{1jk}(2x_j - 1) = 0 & (\forall k \in [\ell]), \\ 0 \leq x_j \leq 1 & (\forall j \in [m]) \end{array} \right\}.$$

A feasible point $x \in P$ is a simultaneously envy-free fractional allocation, where x_j is interpreted as the fraction of item g_j allocated to agent 1. Note that, since $x = (1/2, 1/2, \ldots, 1/2)$ is in the polytope, so P is not empty.

Since P is m-dimensional, any extreme point is determined by m linearly independent tight constraints. In particular, at least $m - \ell$ of these tight constraints must be of the form $x_j = 0$ or $x_j = 1$. This implies that every extreme point has at least $m - \ell$ coordinates that are integral (i.e., equal to 0 or 1). Equivalently, there can be at most ℓ fractional coordinates.

We now construct an allocation from an extreme point x^* of P. Note that an extreme point must exist since P is nonempty and bounded. Define

$$I_1 \coloneqq \{g_j \in M : x^*_j = 1\}, \qquad I_2 \coloneqq \{g_j \in M : x^*_j = 0\},$$
$$F_1 \coloneqq \{g_j \in M : 1/2 \leq x^*_j < 1\}, \qquad F_2 \coloneqq \{g_j \in M : 0 < x^*_j < 1/2\}.$$

Then, since all fractional coordinates of x^* lie in $F_1 \cup F_2$, we have $|F_1| + |F_2| \leq \ell$.

Define the allocation $\mathbf{A} = (A_1, A_2)$ by $A_1 = I_1 \cup F_1$ and $A_2 = I_2 \cup F_2$. Since $x^* \in P$, for every dimension $k \in [\ell]$ we have

$$\begin{aligned} v_1(A_1)_k = \sum_{g_j \in A_1} v_{1jk} &\geq \sum_{g_j \in A_1} v_{1jk}(2x^*_j - 1) \\ &= \sum_{g_j \in A_2} v_{1jk}(1 - 2x^*_j) \geq \sum_{g_j \in I_2} v_{1jk} = v_1(A_2 \setminus F_2)_k. \end{aligned}$$

Similarly, by $x^* \in P$, for every dimension $k \in [\ell]$ we have

$$\begin{aligned} v_2(A_2)_k = \sum_{g_j \in A_2} v_{2jk} &\geq \sum_{g_j \in A_2} v_{2jk}(1 - 2x^*_j) \\ &= \sum_{g_j \in A_1} v_{2jk}(2x^*_j - 1) \geq \sum_{g_j \in I_1} v_{2jk} = v_2(A_1 \setminus F_1)_k. \end{aligned}$$

Thus, any envy can be eliminated by removing all the fractional items from the bundle of the other agent. Hence, the allocation is strong sEFℓ. Since $c \geq \ell$, the allocation is strong sEFc as well.

Finally, since the polytope P has polynomial number of constraints and variables, a basic optimal solution can be computed in polynomial time via standard linear programming techniques. Hence, the resulting allocation $\mathbf{A}$ can be found in polynomial time. ◀

▶ **Theorem 15.** *Checking whether a given allocation is strong sEFc is strongly NP-complete, even for the identical binary case with two agents, when c is part of the input.*

Proof. We first note that the problem is in NP. Indeed, for any distinct agents i, i' and any candidate removal set $X \subseteq A_{i'}$ with $|X| \leq c$, we can verify in polynomial time that for every dimension $k \in L$, it holds that $v_i(A_i)_k \geq v_i(A_{i'} \setminus X)_k$.

We now prove strong NP-hardness by reduction from the *3-Dimensional Matching (3DM)* problem, which is known to be strongly NP-complete [16]. An instance of 3DM consists of three disjoint sets $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, $Z = \{z_1, \ldots, z_n\}$, and a set $T \subseteq X \times Y \times Z$ of triplets. The goal is to decide whether there exists a perfect matching $T' \subseteq T$ of size n, i.e., every element of $X \cup Y \cup Z$ appears in exactly one triple in T'. Without loss of generality, we assume that $|T| > n$ since otherwise it is easy to verify existence of a perfect matching.

Given an instance of the 3DM problem, we construct an instance of the MDFA problem $(N, M, L, (v_i)_{i\in N})$ with an allocation $\mathbf{A}$ and a positive integer c as follows. Let $N = \{1, 2\}$, $M = T \cup \{g^*\}$ (i.e., one item from each triplet in T, plus a special item g^*), and $L = X \cup Y \cup Z$. For every agent $i \in N$, item $t = (x, y, z) \in T$ $(= M \setminus \{g^*\})$, and dimension $w \in L$, define

$$v_i(t)_w = \begin{cases} 1 & \text{if } w \in \{x, y, z\}, \\ 0 & \text{otherwise.} \end{cases}$$

For the special item g^*, set $v_i(g^*)_w = 1$ for all $i \in N$ and $w \in L$. Since the valuation functions are defined in the same way for both agents, the instance is identical and binary. In addition, we define an allocation $\mathbf{A} = (\{g^*\}, M \setminus \{g^*\})$, and we set the parameter $c = |T| - n$ (≥ 1).

We now show that $\mathbf{A}$ is strong sEFc if and only if the 3DM instance is a yes-instance.

(⇒) Suppose that $\mathbf{A}$ is strong sEFc. By definition, we must have

$$1 = v_1(A_1)_w \geq v_1(A_2 \setminus Q)_w \qquad (\forall w \in L)$$

for some $Q \subseteq A_2$ with $|Q| \leq c = |T| - n$. Define $T' = T \setminus Q$. Since $|Q| \leq c = |T| - n$, it follows that $|T'| = |T| - |Q| \geq n$. Now, we prove that $T' = T \setminus Q$ is a perfect matching for the 3DM.

Suppose, for contradiction, that T' is not a perfect matching. Then, since $|T'| \geq n$, there exists $w^* \in X \cup Y \cup Z$ that appears at least twice in T'. However, this is impossible because

$$1 = v_1(A_1)_{w^*} \geq v_1(A_2 \setminus Q)_{w^*} = \left|\left\{t = (x, y, z) \in A_2 \setminus Q : w \in \{x, y, z\}\right\}\right| \geq 2.$$

Thus, every $w \in X \cup Y \cup Z$ appears at most once in T'. Since $|T'| \geq n$, every element $w \in X \cup Y \cup Z$ appears exactly once in T'. In other words, T' is a perfect matching.

(⇐) Conversely, suppose that the given 3DM instance is a yes-instance; that is, there is a perfect matching $T' \subseteq T$. Define the removal set $Q = T \setminus T'$. Then, we have

$$v_1(A_1)_w = 1 = \left|\left\{t = (x, y, z) \in T' : w \in \{x, y, z\}\right\}\right| = v_1(T')_w = v_1(A_2 \setminus Q)_w$$

for any $w \in L$. Thus, the envy from agent 1 to agent 2 can be eliminated by removing $|Q| = |T| - n = c$ items. In addition, agent 2 does not envy agent 1 because

$$v_2(A_2)_w \geq v_2(T')_w = \left|\left\{t = (x, y, z) \in T' : w \in \{x, y, z\}\right\}\right| = 1 = v_2(A_1)$$

for any $w \in L$. Therefore, $\mathbf{A}$ is a strong sEFc allocation.

Since the reduction can be performed in polynomial time and $\mathbf{A}$ is strong sEFc if and only if the 3DM instance is a yes-instance, checking whether a given allocation is strong sEFc is strongly NP-hard even for the identical binary case. ◀

Degrees of Second and Higher-Order Polynomials

Donghyun Lim
KAIST, Daejeon, Republic of Korea

Martin Ziegler
KAIST, Daejeon, Republic of Korea

Abstract

Second-order polynomials generalize classical (=first-order) ones in allowing for additional variables that range over functions rather than values. We are motivated by their applications in higher-order computational complexity theory, extending for instance discrete classes (like P/FP or PSPACE/FPSPACE) to operators in Analysis [Kapron&Cook'96], [Kawamura&Cook'12]. The degree subclassifies ordinary polynomial growth into linear, quadratic, cubic, etc. To similarly classify second-order polynomials, we (well-)define their degree by structural induction as an "arctic" first-order polynomial: a term/expression over integer variable $\mathbf{D}$ and operations $+$ and $\cdot$ *and* binary max(). This generalized degree turns out to transform nicely under (now two kinds of) polynomial composition. As examples, we collect and determine the degrees of previous and new asymptotic analyses of algorithms and operators receiving function/oracle arguments. Then we motivate and introduce third-order polynomials and their degrees as arctic second-order polynomials, along with their transformations under three kinds of composition. Proceeding to fourth order and beyond yields a hierarchy, with characterization in Simply Typed Lambda Calculus.

2012 ACM Subject Classification Mathematics of computing; Theory of computation → Higher order logic; Theory of computation → Type theory

Keywords and phrases Logic in Computer Science, Higher Order Program Analysis, Asymptotic Type Theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.42

Related Version *arXiv Version*: http://arXiv.org/abs/2305.03439 [21]

Funding This work was supported by the National Research Foundation of Korea and by the Korean Ministry of Science and ICT with grants NRF-2017R1E1A1A03071032 and NRF-2016K1A3A7A03950702 and by KAIST in-house grant.

Acknowledgements We thank anonymous reviewers for feedback and guidance.

1 Motivating Examples and Overview

Polynomial – as opposed to, say, exponential – growth is investigated in areas such as Chemistry (reaction kinetics) and Mathematics (Gromov's theorem) and of course Computer Science (Cobham–Edmonds Thesis). The *degree* of polynomial growth provides a refined classification into linear, quadratic, cubic, quartic, quintic, etc. We focus on univariate polynomials over $\mathbb{N}$. For example $15\,\mathbf{D}^3+2\,\mathbf{D}+4$ is of degree 3 in variable $\mathbf{D}$.

So-called second-order polynomials, involving an additional variable ranging over *functions* on natural numbers (instead of values, *i.e.*, one step up the type hierarchy), arise naturally in many areas: including but not limited to characterizations of computational complexity classes and reductions on higher types [30, 13, 15, 7, 25, 2].

▶ **Example 1.** $\mathbf{\Phi}\Big(\mathbf{\Phi}^3\big(\mathbf{\Phi}^5(\mathbf{N})\big)\Big)\cdot\big(\mathbf{\Phi}(\mathbf{N}^2)+\mathbf{N}^9\big)\cdot\mathbf{N}^4\ +$
$+\ \mathbf{N}^{999}\cdot\mathbf{\Phi}\big(3\,\mathbf{N}^5+4\mathbf{\Phi}^8(\mathbf{N}+2)\cdot\mathbf{\Phi}(7\,\mathbf{N})+\mathbf{\Phi}^6(1)\big)\ +\ \mathbf{\Phi}^{50}(\mathbf{N}^9)$
is a *second*-order polynomial $\mathbf{\Pi}$ in first-order variable $\mathbf{N}$ and second-order variable $\mathbf{\Phi}$.

© Donghyun Lim and Martin Ziegler;
licensed under Creative Commons License CC-BY 4.0

The *degree* of this second-order polynomial turns out to be the following expression $\widetilde{\mathbf{P}}(\mathbf{D}) =$

$$\max\left\{\ \mathbf{D}\cdot(3\,\mathbf{D})\cdot(5\,\mathbf{D})+\max\{2\,\mathbf{D},9\}+4\ ,\ 999+\mathbf{D}\cdot\max\{5,8\,\mathbf{D}+\mathbf{D}\}\ ,\ 450\cdot\mathbf{D}\ \right\} \quad (1)$$

Other than classical polynomials, it involves the (binary) max operation and is thus called "arctic": see Remark 2a) below. However, whenever $\mathbf{D}$ has value $d \geq 6$, Expression (1) semantically coincides with the above simple cubic polynomial $15\,\mathbf{D}^3+2\,\mathbf{D}+4$. ⌟

In Section 2 below, Definition 5 formally recalls second-order polynomials; and Subsection 2.3 collects both previous and new examples and applications from Complexity Theory and Analysis. In Subsection 2.1, Definition 9 introduces the notion of second-order polynomial degree; and Proposition 12 records some of its properties as well as connections to ordinary (=first-order) polynomial degrees. Theorem 7 asserts semantic completeness.

Section 3 climbs further up the type hierarchy: first to third-order polynomials, whose degrees are arctic second-order polynomials (Subsection 3.2); and then (Subsection 3.3) to the higher-order cases, including a characterization in terms of a certain fragment of Simply Typed Lambda Calculus in Subsection 3.4. Proofs are deferred to Appendices A and B; those omitted due to page constraints can be found in [21].

Our notions of higher-order degrees offer sub-classifications of higher-degree polynomial growth to various levels of detail; cf. Table 2 in Subsection 2.1. Such a sub-classification had been envisioned/requested by Akitoshi Kawamura (personal conversation, ca. 2015).

▶ Remark 2.

a) "Tropical" historically refers to expressions over $(\min,+,\infty,0)$ instead of classical $(+,\cdot,0,1)$ [26], "arctic" dually to $(\max,+,-\infty,0)$: used in Definition 3 below. Both are instances of so-called "exotic" semi-rings [5, §2].

b) First-order polynomials are commonly defined syntactically as a family of well-formed expressions over $+$ and $\cdot$ and 1: over one or several ($M \in \mathbb{N}$) variables $\vec{\mathbf{X}} = (X_1,\ldots,X_M)$, with or without (="pure") additional constants from some semi/ring $\mathcal{R}$, with or without (="positive") subtraction. Logically speaking, they are precisely the elements of a suitable term language. Each syntactic polynomial $\mathbf{P} \in \mathcal{R}[\vec{\mathbf{X}}]$ gives rise semantically to a unique total function $\overline{\mathbf{P}} = p : \mathcal{R}^M \to \mathcal{R}$.

c) Regarding the converse direction – namely translating certain functions (semantics) to syntactic polynomials – existence and uniqueness questions can be challenging depending on the particular setting. [28] for example investigates which multivariate functions over a ring $\mathcal{R}$ *with* zero-divisors can be represented as polynomials at all. And Hilbert's famous *Nullstellensatz* characterizes the *non*-uniqueness of polynomial representations of a multivariate partial function, defined on some algebraic variety over $\mathcal{R} = \mathbb{C}$.

d) By virtue of Commutative and Associative and Distributive Laws, any classical multivariate polynomial can be rewritten – see Equation (3) – as a linear combination of monomials $\mathbf{X}_1^{d_1}\cdots\mathbf{X}_M^{d_M}$ of lexicographically strictly increasing multidegrees $(d_1,\ldots,d_M)$. And for $\mathcal{R} = \mathbb{N}$, this syntactic normal form is also semantically unique:

e) Let $\mathbf{P}(\vec{\mathbf{X}}), \mathbf{Q}(\vec{\mathbf{X}})$ denote two ordinary syntactically non-equivalent (Equation 3) positive pure polynomials in variables $\vec{\mathbf{X}} = (\mathbf{X}_1,\ldots,\mathbf{X}_M)$ over $\mathbb{N}$. Then there exists an assignment $\vec{x} = (x_m)_m \in \mathbb{N}^M$ that makes them evaluate differently: $\overline{\mathbf{P}}(\vec{x}) \neq \overline{\mathbf{Q}}(\vec{x})$.

Item e) follows for instance from the (DeMillo-Lipton-) Schwartz-Zippel Lemma, based on monomial normal form. Theorem 7 generalizes Item e) to the second-order case.

Introducing second-order polynomial degrees had been attempted in [31]: a brief note summarizing a spontaneous short Dagstuhl idea with*out* (explicit) definitions nor proofs, which furthermore wrongly claims that the second-order degree is a (non-arctic) polynomial –

rendering the remaining claims moot. Section 2 puts said ideas from [31] on a sound foundation with definitions and proofs and elaborate examples; and Section 3 takes a new perspective on the first and second-order case, in order to then generalize to third and higher-order polynomials and degrees.

Table 1 List of symbols and notational conventions.

$\mathbf{N}, \mathbf{M}, \mathbf{D}$	first-order variables
$\overline{\mathbf{N}} = n, \overline{\mathbf{M}} = m, \overline{\mathbf{D}} = d \in \mathbb{N}$	values of said variables
$\mathbf{P} = \mathbf{P}(\vec{\mathbf{N}}), \mathbf{Q} = \mathbf{Q}(\mathbf{M})$	pure positive multi/uni-variate 1st-order polynomials
$\overline{\mathbf{P}} = p, \overline{\mathbf{Q}} = q : \mathbb{N}\nearrow\mathbb{N}$	induced monotonic functions
$d = \deg(\mathbf{P}) \in \mathbb{N}$	(total) degree of non-zero first-order polynomial
$\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}}$	arctic first-order polynomials
$\mathbf{Q} = \lim \widetilde{\mathbf{Q}}$	induced asymptotic first-order polynomial
$\mathbf{\Phi}, \mathbf{\Psi}$	second-order variables
$\overline{\mathbf{\Phi}} = \phi, \overline{\mathbf{\Psi}} = \psi : \mathbb{N}\nearrow\mathbb{N}$	values of said variables
$\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \vec{\mathbf{N}}), \mathbf{\Xi}(\mathbf{\Psi}, \mathbf{N})$	(multi/uni-variate) second-order polynomials
$\overline{\mathbf{\Pi}} : (\mathbb{N}\nearrow\mathbb{N}) \times \mathbb{N}^m \nearrow \mathbb{N}$	induced monotonic mixed functionals
$\overline{\mathbf{\Xi}} = \mathcal{Q} : (\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})$	Curry-ed monotonic operator
$\widetilde{\mathbf{P}} = \mathrm{Deg}(\mathbf{\Pi})$	arctic first-order polynomial as second-order degree
$\mathbf{P} = \lim \mathrm{Deg}(\mathbf{\Pi})$	1st-order polynomial as asymptotic 2nd-order degree
$d = \deg\left(\lim \mathrm{Deg}(\mathbf{\Pi})\right) \in \mathbb{N}$	nesting depth of second-order polynomial $\mathbf{\Pi}$
$\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}}$	arctic second-order polynomials
$\mathfrak{F}, \mathfrak{G}$	third-order variables
$\overline{\mathfrak{F}} = \mathcal{F}, \overline{\mathfrak{G}} = \mathcal{G} : (\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})$	values of said variables
$\mathfrak{P} = \mathfrak{P}(\mathfrak{F}, \mathbf{\Phi}, \mathbf{N}), \mathfrak{Q}(\mathfrak{G}, \mathbf{\Psi}, \mathbf{M})$	third-order polynomials
$\overline{\mathfrak{P}} : \left((\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})\right) \times (\mathbb{N}\nearrow\mathbb{N}) \times \mathbb{N} \nearrow \mathbb{N}$	induced monotonic mixed hyper-functional
$\overline{\mathfrak{P}} : \left((\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})\right) \times (\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})$	Curry-ed monotonic mixed operator
$\overline{\mathfrak{P}} : \left((\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})\right) \nearrow \left((\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})\right)$	Curry-ed monotonic hyper-operator
$\widetilde{\mathbf{\Pi}} = \mathrm{DEG}(\mathfrak{P})$	arctic second-order polynomial as third-order degree

1.1 Arctic First-Order Polynomials

Definition 9 below defines the degree $\mathrm{Deg}(\mathbf{\Pi})$ of a second-order polynomial to be an ordinary polynomial, but one involving $\max()$.

▶ **Definition 3.** *An* arctic *(first-order) polynomial $\widetilde{\boldsymbol{P}}(\vec{\boldsymbol{X}})$ in variables $\vec{\boldsymbol{X}}$ is a well-formed expression over $\boldsymbol{X}$ and constant symbols $0, 1$ (unary) and binary $+, \cdot, \max()$.*

Recall Example 1 and Remark 2a). As opposed to Definition 5, the constant 0 is permitted here. We refrain from spelling out the obvious semantics $\overline{\mathbf{P}}$ of an arctic polynomial $\widetilde{\mathbf{P}}$. Note that, when evaluating $\widetilde{\mathbf{P}}(\vec{x})$ on an arbitrary fixed integer vector $\vec{x}$, any occurrence of $\max()$ evaluates to at least one of its two arguments. Moreover, in the univariate case $\vec{X} = X$ and as $\overline{X} \to \infty$, the role of the dominant argument in $\max()$ may switch only finitely often, as one can see by structural induction:

▶ **Lemma 4.**

a) *For any two distinct univariate positive ordinary polynomials $\boldsymbol{P} = \boldsymbol{P}(\boldsymbol{N})$ and $\boldsymbol{Q} = \boldsymbol{Q}(\boldsymbol{N})$, their associated polynomial functions $p = \overline{\boldsymbol{P}}$ and $q = \overline{\boldsymbol{Q}}$ satisfy*

$$\exists N \in \mathbb{N} : \quad \big(\forall n \geq N : \ p(n) > q(n)\big) \quad \vee \quad \big(\forall n \geq N : \ p(n) < q(n)\big) \ . \tag{2}$$

b) *Fix univariate arctic polynomial $\widetilde{\boldsymbol{P}} = \widetilde{\boldsymbol{P}}(\boldsymbol{D})$. Its value coincides, for all sufficiently large arguments $d \in \mathbb{N}$, with $\overline{\boldsymbol{P}}(d)$ for a unique* ordinary *(positive pure) polynomial $\boldsymbol{P} = \boldsymbol{P}(\boldsymbol{D})$.*

Call $\mathbf{P} \in \mathbb{N}[\mathbf{D}]$ from Lemma 4b) the *asymptotic polynomial* induced by univariate arctic $\widetilde{\mathbf{P}}$, written $\mathbf{P} = \lim \widetilde{\mathbf{P}}$. There is no risk of confusing the polynomial $\lim \widetilde{\mathbf{P}}$ with the "value" $\infty = \lim_{d\to\infty} \widetilde{\mathbf{P}}(d)$. We record that $\lim(\widetilde{\mathbf{P}} + \widetilde{\mathbf{Q}}) = (\lim \widetilde{\mathbf{P}}) + (\lim \widetilde{\mathbf{Q}})$ and $\lim(\widetilde{\mathbf{P}} \cdot \widetilde{\mathbf{Q}}) = (\lim \widetilde{\mathbf{P}}) \cdot (\lim \widetilde{\mathbf{Q}})$. Lemma 4 is limited to the univariate case: multivariate arctic terms like $\max(\mathbf{X}\,\mathbf{Y}^2, \mathbf{X}^2\,\mathbf{Y})$ may not asymptotically coincide with an ordinary one; see also Example 19.

2 Second-Order Polynomials

We consider "positive pure" (univariate) *second*-order polynomials, that is, with syntactically well-formed expressions involving – in addition to operations $+, \cdot$ and 1 as only constant and classical variables $\vec{\mathbf{N}} = (\mathbf{N}_1, \ldots, \mathbf{N}_M)$ – some unary function-type variable $\mathbf{\Phi}$ [13]:

▶ **Definition 5.** *Second-order polynomials, like* $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \vec{\boldsymbol{N}})$ *and* $\mathbf{\Xi} = \mathbf{\Xi}(\mathbf{\Phi}, \vec{\boldsymbol{N}})$*, are syntactically generated by the Backus-Naur rules*

$$\mathbf{\Pi}, \mathbf{\Xi} \quad ::= \quad 1 \mid \boldsymbol{N}_1 \mid \ldots \mid \boldsymbol{N}_M \mid \mathbf{\Pi} + \mathbf{\Xi} \mid \mathbf{\Pi} \cdot \mathbf{\Xi} \mid \mathbf{\Phi}(\mathbf{\Pi}) \ .$$

In other words, second-order polynomials are the least class of formal expressions (=terms)

- *that include constant* 1 *and variables* $(\boldsymbol{N}_1, \ldots, \boldsymbol{N}_M) = \vec{\boldsymbol{N}}$
- *and are closed under binary addition* $+$ *and product* $\cdot$
- *Moreover, when* $\mathbf{\Pi}$ *is a second-order polynomial, then so is* $\mathbf{\Phi}(\mathbf{\Pi})$*:*

Regarding semantics, we continue using overline to denote the interpretation of an expression as a mapping: Each variable $\boldsymbol{N}$ *may take values* $\overline{\boldsymbol{N}} = n$ *ranging over* $\mathbb{N}$*; and* $\overline{\mathbf{\Phi}}$ *ranges over* $(\mathbb{N}\nearrow\mathbb{N})$*, the set of nondecreasing total unary functions* $\phi : \mathbb{N} \nearrow \mathbb{N}$*. Extended by structural induction to compound expressions, let* $\overline{\mathbf{\Phi}(\mathbf{\Pi})}$ *evaluate to* $\phi\big(\overline{\mathbf{\Pi}}(\phi, n)\big)$ *and* $\overline{1} = 1$*.*

Recall Example 1, note that we prohibit the constant 0. Exponentiation (only to a natural number power) abbreviates repeated multiplication: $\mathbf{N}^3 = \mathbf{N} \cdot \mathbf{N} \cdot \mathbf{N}$ and $\mathbf{\Phi}^2(\mathbf{\Xi}) = \mathbf{\Phi}(\mathbf{\Xi}) \cdot \mathbf{\Phi}(\mathbf{\Xi})$, as opposed to repeated composition $\mathbf{\Phi} \circ \mathbf{\Phi} = \mathbf{\Phi}^{(2)}$. Commutative, associative, distributive laws extend from numbers to (both first and) second-order polynomials pointwise:

▶ **Definition 6.** *Let "*$\approxeq$*" denote syntactic equivalence of terms up to Associative, Commutative, and Distributive Laws; formally the equivalence relation generated by the following rules:*

$$\mathbf{\Pi} + \mathbf{\Xi} \approxeq \mathbf{\Xi} + \mathbf{\Pi}, \qquad (\mathbf{\Pi} + \mathbf{\Xi}) + \mathbf{\Lambda} \approxeq \mathbf{\Pi} + (\mathbf{\Xi} + \mathbf{\Lambda}), \tag{3}$$
$$\mathbf{\Pi} \cdot \mathbf{\Xi} \approxeq \mathbf{\Xi} \cdot \mathbf{\Pi}, \quad (\mathbf{\Pi} \cdot \mathbf{\Xi}) \cdot \mathbf{\Lambda} \approxeq \mathbf{\Pi} \cdot (\mathbf{\Xi} \cdot \mathbf{\Lambda}), \quad \mathbf{\Pi} \cdot 1 \approxeq \mathbf{\Pi}, \quad \mathbf{\Pi} \cdot (\mathbf{\Xi} + \mathbf{\Lambda}) \approxeq \mathbf{\Pi} \cdot \mathbf{\Xi} + \mathbf{\Pi} \cdot \mathbf{\Lambda}$$
$$\big(\mathbf{\Pi} \approxeq \mathbf{\Pi}' \ \wedge \ \mathbf{\Xi} \approxeq \mathbf{\Xi}'\big) \quad \Rightarrow \quad \mathbf{\Pi} + \mathbf{\Xi} \approxeq \mathbf{\Pi}' + \mathbf{\Xi}' \ \wedge \ \mathbf{\Pi} \cdot \mathbf{\Xi} \approxeq \mathbf{\Pi}' \cdot \mathbf{\Xi}' \ \wedge \ \mathbf{\Phi}(\mathbf{\Pi}) \approxeq \mathbf{\Phi}(\mathbf{\Pi}')$$

Syntactic equivalence is semantically *sound* for 2nd-order polynomials: $\mathbf{\Pi} \approxeq \mathbf{\Xi} \Rightarrow \overline{\mathbf{\Pi}} \equiv \overline{\mathbf{\Xi}}$. Generalizing Remark 2e), the converse – semantic *completeness* – is also true, but less obvious:

▶ **Theorem 7.** *Let* $\mathbf{\Pi}(\mathbf{\Phi}, \boldsymbol{N}), \mathbf{\Xi}(\mathbf{\Phi}, \boldsymbol{N})$ *be syntactically non-equivalent second-order polynomials. There exists an assignment* $n \in \mathbb{N}$ *and* $\phi \in (\mathbb{N}\nearrow\mathbb{N})$ *such that* $\overline{\mathbf{\Pi}}(\phi, n) \neq \overline{\mathbf{\Xi}}(\phi, n)$*.*

See [20, 21] for a proof, omitted here due to space limitations. A similar result has been shown for Simply Typed Lambda Calculus (see also Subsection 3.4 below) modulo integer arithmetic [27, Theorems 5.2+5.6], with two caveats: function variable $\mathbf{\Phi}$ is supposed bi-variate [27, p.684] and runs over all (not just monotonic) total integer mappings [27, Definition 4.11].

2.1 Second-Order Polynomial Degree

The total degree $\deg(\mathbf{P}) \in \mathbb{N}$ of a classical multivariate polynomial $\mathbf{P} = \mathbf{P}(\vec{\mathbf{X}}) \neq 0$ is commonly defined first for monomials, and then for linear combinations of the latter: relying on the monomial normal form, recall Remark 2d+e). An alternative equivalent definition proceeds by structural induction, for instance for positive pure polynomials as follows:

$$\begin{aligned} &\deg(1) = 0, \qquad \deg(\mathbf{P}+\mathbf{Q}) = \max\{\deg(\mathbf{P}), \deg(\mathbf{Q})\}, \\ &\deg(\mathbf{X}_m) = 1, \qquad \deg(\mathbf{P}\cdot\mathbf{Q}) = \deg(\mathbf{P}) + \deg(\mathbf{Q}) \ . \end{aligned} \tag{4}$$

The former approach builds on monomial normal form (Remark 2d) while the latter needs to separately establish well-definition, namely invariance under syntactic equivalence (3):

$$\mathbf{P} \approxeq \mathbf{Q} \quad \Rightarrow \quad \deg(\mathbf{P}) \ = \ \deg(\mathbf{Q}) \ , \tag{5}$$

which we generalize in Remark 10 below. Either way, the *Rule of Composition* then follows, in the univariate case expressed concisely as: $\deg(\mathbf{P} \circ \mathbf{Q}) = \deg(\mathbf{P}) \cdot \deg(\mathbf{Q})$.

▶ Remark 8. Strictly speaking, $\mathbf{P} \circ \mathbf{Q}$ needs to be defined syntactically: for instance by structural induction on $\mathbf{P}$, essentially replacing each occurrence of variable $\mathbf{X}$ in $\mathbf{P}$ with $\mathbf{Q}$. And said inductive definition then gets justified semantically by concluding $\overline{\mathbf{P} \circ \mathbf{Q}} = \overline{\mathbf{P}} \circ \overline{\mathbf{Q}}$, where (only) the right-hand side means composition of functions. Definition 11 and Proposition 12 below proceed in this very way, for second-order polynomials.

For univariate polynomials, the integer total order on degrees "$\deg(\mathbf{P}) \leq \deg(\mathbf{Q})$" captures Landau's pre-order of asymptotic growth "$\mathbf{P}(\mathbf{N}) = \mathcal{O}\big(\mathbf{Q}(\mathbf{N})\big)$"; recall Lemma 4a). This suggests extending Equation (4) from ordinary to second-order polynomials:

▶ **Definition 9.** *The (second-order) degree of a second-order polynomial* $\mathbf{\Pi}$ *in* $\mathbf{\Phi}$ *and* $\vec{\boldsymbol{N}} = (\boldsymbol{N}_1, \ldots, \boldsymbol{N}_m)$ *is an arctic polynomial* $\mathrm{Deg}\big(\mathbf{\Pi}\big)$ *in* $\boldsymbol{D}$*, given inductively by*

$$\begin{aligned} &\mathrm{Deg}(1) \ := \ 0, \quad \mathrm{Deg}(\boldsymbol{N}_m) \ := \ 1, \qquad \mathrm{Deg}(\mathbf{\Pi}+\mathbf{\Xi}) \ := \ \max\{\mathrm{Deg}(\mathbf{\Pi}), \mathrm{Deg}(\mathbf{\Xi})\}, \\ &\mathrm{Deg}(P \cdot Q) \ := \ \mathrm{Deg}(P) + \mathrm{Deg}(Q), \quad \textit{and} \quad \mathrm{Deg}\big(\mathbf{\Phi}(\mathbf{\Pi})\big) \ := \ \boldsymbol{D} \cdot \mathrm{Deg}(\mathbf{\Pi}) \ . \end{aligned} \tag{6}$$

We write "$\boldsymbol{D} = \mathrm{Deg}(\mathbf{\Phi})$*" to emphasize that first-order variable* $\boldsymbol{D}$ *in* $\mathrm{Deg}(\mathbf{\Pi})$ *corresponds to second-order argument* $\mathbf{\Phi}$ *in* $\mathbf{\Pi}$*, particularly when later generalizing to several second-order variables; see Remark 13b+e).*

Related work [16] had investigated *linear* second-order polynomials: defined by omitting/-prohibiting multiplication from Definition 5 (but still allowing for addition $+$ and nesting $\mathbf{\Phi}$). These are precisely those having as degree an arctic polynomial with*out* addition $+$ (but still with multiplication $\cdot$ and max); recall that Definition 5 allows only the constant 1.

▶ Remark 10. Definition 9 is well-defined, namely it respects syntactic equivalence (3): Arithmetical commutativity and associativity of $+$ and $\cdot$ translate to "arctic" commutativity and associativity of max() and $+$, respectively; multiplication by 1 translates to addition of 0; and distributivity $\mathbf{\Pi} \cdot (\mathbf{\Xi} + \mathbf{\Lambda}) \approxeq \mathbf{\Pi} \cdot \mathbf{\Xi} + \mathbf{\Pi} \cdot \mathbf{\Lambda}$ translates to

$$\mathrm{Deg}(\mathbf{\Pi}) + \max\big\{\,\mathrm{Deg}(\mathbf{\Xi}), \mathrm{Deg}(\mathbf{\Lambda})\big\} \ = \ \max\big\{\,\mathrm{Deg}(\mathbf{\Pi}) + \mathrm{Deg}(\mathbf{\Xi}) \,,\, \mathrm{Deg}(\mathbf{\Pi}) + \mathrm{Deg}(\mathbf{\Lambda})\big\} \ . \tag{7}$$

Reflecting that every ordinary polynomial is *a fortiori* also a second-order polynomial, Definition 9 extends the classical degree: $\mathrm{Deg}(\mathbf{P}) = \deg(\mathbf{P}) \in \mathbb{N}$ for any (positive pure) multivariate first-order polynomial $\mathbf{P} \neq 0$.

According to Lemma 4b), the degree $\mathrm{Deg}(\mathbf{\Pi})$ of a second-order polynomial induces an ordinary univariate polynomial $\lim \mathrm{Deg}(\mathbf{\Pi})$ – which we call $\mathbf{\Pi}$'s *asymptotic* degree.

Table 2 Stating "second-order polynomial growth" in various levels of detail.

Growing as ...	Example
a given second-order polynomial $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$	(complicated) $\mathbf{\Pi}$ from Example 1
(some *un*specified second-order polynomial) having a *given* arctic first-order polynomial as degree	(simpler) $\widetilde{\mathbf{P}}(\mathbf{D})$ from Example 1
(some *un*specified second-order polynomial) having a given first-order polynomial as *asymptotic* degree	"$15\,\mathbf{D}^3 + 2\,\mathbf{D} + 4$" (Example 1)
(some *un*specified second-order polynomial) having a given nesting depth according to Proposition 12d)	"$3 \in \mathbb{N}$" in Example 1
some *un*specified second order polynomial	[15, Definition 3.2]

2.2 Second-Order Polynomial Compositions

Second-order polynomials naturally compose in *two* different ways, here denoted $\circ$ and $\star$:

▶ **Definition 11.** *Let* $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \boldsymbol{N})$ *and* $\mathbf{\Xi} = \mathbf{\Xi}(\mathbf{\Phi}, \boldsymbol{N})$ *be univariate second-order polynomials. Define their compositions* $\mathbf{\Pi} \circ \mathbf{\Xi}$ *and* $\mathbf{\Pi} \star \mathbf{\Xi}$ *by structural induction as follows:*

$$
\begin{aligned}
1 \circ \mathbf{\Xi} &:= 1, \qquad 1 \star \mathbf{\Xi} := 1, & \boldsymbol{N} \circ \mathbf{\Xi} &:= \mathbf{\Xi}, \qquad \boldsymbol{N} \star \mathbf{\Xi} := \boldsymbol{N}, \\
(\mathbf{\Pi}_1 + \mathbf{\Pi}_2) \circ \mathbf{\Xi} &:= (\mathbf{\Pi}_1 \circ \mathbf{\Xi}) + (\mathbf{\Pi}_2 \circ \mathbf{\Xi}), & (\mathbf{\Pi}_1 + \mathbf{\Pi}_2) \star \mathbf{\Xi} &:= (\mathbf{\Pi}_1 \star \mathbf{\Xi}) + (\mathbf{\Pi}_2 \star \mathbf{\Xi}) \\
(\mathbf{\Pi}_1 \cdot \mathbf{\Pi}_2) \circ \mathbf{\Xi} &:= (\mathbf{\Pi}_1 \circ \mathbf{\Xi}) \cdot (\mathbf{\Pi}_2 \circ \mathbf{\Xi}), & (\mathbf{\Pi}_1 \cdot \mathbf{\Pi}_2) \star \mathbf{\Xi} &:= (\mathbf{\Pi}_1 \star \mathbf{\Xi}) \cdot (\mathbf{\Pi}_2 \star \mathbf{\Xi}) \\
\mathbf{\Phi}(\mathbf{\Pi}) \circ \mathbf{\Xi} &:= \mathbf{\Phi}(\mathbf{\Pi} \circ \mathbf{\Xi}), & \mathbf{\Phi}(\mathbf{\Pi}) \star \mathbf{\Xi} &:= \mathbf{\Xi} \circ (\mathbf{\Pi} \star \mathbf{\Xi})
\end{aligned}
$$

$\mathbf{\Pi}\big(\mathbf{\Phi}, \mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})\big) := \big(\mathbf{\Pi} \circ \mathbf{\Xi}\big)(\mathbf{\Phi}, \mathbf{N})$ essentially replaces in $\mathbf{\Pi}$ every occurrence of first-order variable $\mathbf{N}$ with $\mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})$; and $\mathbf{\Pi}\big(\mathbf{\Xi}(\mathbf{\Phi}, \cdot), \mathbf{N}\big) := \big(\mathbf{\Pi} \star \mathbf{\Xi}\big)(\mathbf{\Phi}, \mathbf{N})$ replaces in $\mathbf{\Pi}$ every occurrence of second-order variable $\mathbf{\Phi}$ with $\mathbf{\Xi}(\mathbf{\Phi}, \cdot)$.
Note that $\mathbf{\Phi}(\mathbf{N}) = \mathbf{\Phi}$; and $\mathbf{P} = \mathbf{P} \circ \mathbf{N}$ for first-order polynomials $\mathbf{P}$: justifying the common notation $\mathbf{P} = \mathbf{P}(\mathbf{N})$. In the second-order case, there is no danger of confusing the notation $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$ with a composition, since the pair $(\mathbf{\Phi}, \mathbf{N})$ is not a single second-order polynomial.

Like in the classical case, the notion of degree translates composition "$\circ$" to multiplication "$\cdot$". The other kind "$\star$" of composition, new to the second-order case, translates as ordinary composition "$\circ$" of first-order (arctic) polynomials:

▶ **Proposition 12.** *Fix univariate second-order polynomials* $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \boldsymbol{N})$ *and* $\mathbf{\Xi} = \mathbf{\Xi}(\mathbf{\Phi}, \boldsymbol{N})$.

a) $\overline{(\mathbf{\Pi} \circ \mathbf{\Xi})}$ *is again a second-order polynomial in* $(\mathbf{\Phi}, \boldsymbol{N})$*, with semantics* $\overline{\big(\mathbf{\Pi} \circ \mathbf{\Xi}\big)}(\phi, n) = \overline{\mathbf{\Pi}}\big(\phi, \overline{\mathbf{\Xi}}(\phi, n)\big)$ *for all* $n \in \mathbb{N}$ *and all* $\phi \in (\mathbb{N} \nearrow \mathbb{N})$*. Furthermore*

$$\mathrm{Deg}\,\big(\mathbf{\Pi} \circ \mathbf{\Xi}\big)(\boldsymbol{D}) \;=\; \mathrm{Deg}\,\big(\mathbf{\Pi}\big)(\boldsymbol{D}) \;\cdot\; \mathrm{Deg}\,\big(\mathbf{\Xi}\big)(\boldsymbol{D}) \;.$$

b) $\big(\mathbf{\Pi} \star \mathbf{\Xi}\big)(\mathbf{\Phi}, \boldsymbol{N})$ *is again a second-order polynomial in* $(\mathbf{\Phi}, \boldsymbol{N})$*, with semantics* $\overline{\mathbf{\Pi}\big(\mathbf{\Xi}(\mathbf{\Phi}, \cdot), \boldsymbol{N}\big)}(\phi, n) = \overline{\mathbf{\Pi}}\big(\overline{\mathbf{\Xi}}(\phi, \cdot), n\big)$ *for* $n \in \mathbb{N}$ *and* $\phi \in (\mathbb{N} \nearrow \mathbb{N})$*. Here* $\overline{\mathbf{\Xi}}(\phi, \cdot) \in (\mathbb{N} \nearrow \mathbb{N})$ *denotes the monotonic mapping* $\lambda n' : \mathbb{N}.\overline{\mathbf{\Xi}}(\phi, n') \in \mathbb{N}$*. Furthermore*

$$\mathrm{Deg}\,\big(\mathbf{\Pi} \star \mathbf{\Xi}\big)(\boldsymbol{D}) \;=\; \big(\mathrm{Deg}(\mathbf{\Pi}) \circ \mathrm{Deg}(\mathbf{\Xi})\big)(\boldsymbol{D}) \;=\; \mathrm{Deg}\,\big(\mathbf{\Pi}\big)\big(\mathrm{Deg}(\mathbf{\Xi})(\boldsymbol{D})\big) \;.$$

c) *Moreover, if* $\mathbf{\Xi}$ *is an ordinary polynomial, then so is* $\mathbf{\Pi} \star \mathbf{\Xi}$*. For* $\mathbf{\Xi}_m \equiv m \in \mathbb{N}$ *the family of constant ordinary polynomials,* $\mathbf{\Pi} \star \mathbf{\Xi}_M$ *is a bivariate polynomial in* (M, N)*.*

d) $\deg\big(\lim \mathrm{Deg}(\mathbf{\Pi})\big) \in \mathbb{N}$ *is well-defined, and coincides with the* (nesting) depth *of* $\mathbf{\Phi}$ *in* $\mathbf{\Pi}$ *according to* [13, Definition 5.9].

The proofs proceed by straight-forward structural induction and are omitted.

▶ Remark 13.

a) Proposition 12a) extends and recovers the first-order case, when $\mathbf{\Pi}$ and $\mathbf{\Xi}$ are ordinary polynomials. When $\mathbf{\Xi} = \mathbf{Q}$ is first-order, then $\mathrm{Deg}(\mathbf{\Pi} \circ \mathbf{\Xi}) = \mathrm{Deg}(\mathbf{\Pi}) \cdot \deg(\mathbf{Q})$: an integer multiple of $\mathrm{Deg}(\mathbf{\Pi})$; analogously when $\mathbf{\Pi}$ is first-order.

b) In Proposition 12c), when $\mathbf{\Xi} = \mathbf{Q}$ is first-order, then $\mathrm{Deg}(\mathbf{\Pi} \star \mathbf{\Xi}) = \mathrm{Deg}\big(\mathbf{\Pi}\big)(\deg \mathbf{Q})$: evaluating the arctic first-order polynomial $\mathrm{Deg}\big(\mathbf{\Pi}\big)(\mathbf{D})$ at the argument $\overline{\mathbf{D}} := \deg(\mathbf{Q}) \in \mathbb{N}$. This semantic coincidence justifies our syntactic convention "$\mathbf{D} = \mathrm{Deg}(\mathbf{\Phi})$" at the end of Definition 9.

c) Regarding Proposition 12d), recall Lemma 4b) that $\lim \mathrm{Deg}(\mathbf{\Pi})$ denotes the unique ordinary polynomial coinciding with arctic $\mathrm{Deg}(\mathbf{\Pi})$ on all sufficiently large arguments. In view of Proposition 12d), second-order polynomial asymptotic growth can be stated with various decreasing degrees of detail and increasing conciseness, as illustrated in Table 2.

d) Regarding the multivariate case, Definition 9 already covers second-order polynomials involving several first-order variables $\vec{\mathbf{N}} = (\mathbf{N}_1, \ldots, \mathbf{N}_M)$. Composition "$\mathbf{\Pi} \circ \mathbf{\Xi}$" then becomes "$\mathbf{\Pi} \circ_m \mathbf{\Xi}$": We refrain from spelling out the inductive replacement of $\mathbf{N}_m$ in $\mathbf{\Pi}$ with $\mathbf{\Xi}$. Proposition 12a) adapts immediately – but of course not Lemma 4.

e) Definition 5 naturally generalizes to several unary second-order variables $\mathbf{\Phi}_m$. The degree (Definition 9) then becomes a *multi*variate arctic first-order polynomial: Replace Equation (6) with $\mathrm{Deg}\big(\mathbf{\Phi}_m(\mathbf{\Pi})\big) := \mathbf{D}_m \cdot \mathrm{Deg}(\mathbf{\Pi})$, where $\mathbf{D}_m = \mathrm{Deg}(\mathbf{\Phi}_m)$; recall Item b). Composition "$\mathbf{\Pi} \star \mathbf{\Xi}$" becomes "$\mathbf{\Pi} \star_m \mathbf{\Xi}$", and Proposition 12b) adapts accordingly.

f) $\mathbf{\Pi} := \mathbf{\Phi}^2(\mathbf{N}^3)$ and $\mathbf{\Xi} := \mathbf{\Phi}(\mathbf{N}^6)$ both have same degree $6\,\mathbf{D}$, but neither $\overline{\mathbf{\Pi}} \leq \mathcal{O}\big(\overline{\mathbf{\Xi}}\big)$ nor $\overline{\mathbf{\Xi}} \leq \mathcal{O}\big(\overline{\mathbf{\Pi}}\big)$; see also Example 19.

2.3 Examples and Applications

Polynomial bounds capture "tame" (as opposed to, say, exponential) growth in many areas:

- In classical Complexity Theory, $\mathbf{P}(\mathbf{N})$ denotes the amount of resources used to process inputs of length $\overline{\mathbf{N}} = n$.
- In Geometric Group Theory, $\mathbf{P}(\mathbf{N})$ denotes the number of distinct group elements expressible by words of length $\overline{\mathbf{N}} = n$ over a fixed symmetric set of generators [9].
- In statistical physics / probability theory, $\mathbf{P}(\mathbf{N})$ denotes the *mixing time* of a dynamical system or stochastic process over a $2^{\mathbf{N}}$ element universe [19].

Generally, establishing polynomially bounded growth is followed by further investigating the (least) *degree* of said polynomial [29, 8, 6]. Recall (before Definition 9) that the integer total order on degrees captures Landau's preorder of asymptotic growth.

The present work enables similarly refined analyses of higher-type problems: Extending the Cobham-Edmonds Thesis, asymptotic growth of a function*al* depending on (first-order $\overline{\mathbf{N}}$ and on) a second-order parameter $\overline{\mathbf{\Phi}}$ is commonly considered "tame" iff it is bounded by (the values of) some second-order polynomial $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$. And its degree $\mathrm{Deg}(\mathbf{\Pi})$ captures, and allows to concisely compare, second-order asymptotic growth; recall Remark 13c) and see Table 2. When its argument $\overline{\mathbf{\Phi}}$ itself grows exponentially, then $\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$ as a *relative* polynomial bound [24, §4] may grow like an exponential tower of height $\deg\big(\lim \mathrm{Deg}(\mathbf{\Pi})\big)$, recall Proposition 12d). We collect here some old and new examples, proofs in Appendix B.

▶ **Example 14** (String Functionals)**.**

i) Suppose traditional Turing machine $\mathcal{M}$ computes function $f : \{0,1\}^* \to \{0,1\}^*$ in polynomial time $\mathbf{P} = \mathbf{P}(\mathbf{M})$ and machine $\mathcal{N}$ computes $g : \{0,1\}^* \to \{0,1\}^*$ in polynomial time $\mathbf{Q} = \mathbf{Q}(\mathbf{N})$. Then executing $\mathcal{M}$ on input $\vec{y} = g(\vec{x})$ of length $m \leq \mathbf{Q}(n)$, obtained

by executing $\mathcal{N}$ on input $\vec{x}$, results in a Turing machine computing $f \circ g$ in total running time $\mathcal{O}\Big(\mathbf{Q}(\mathbf{N}) + \mathbf{P}\big(\mathbf{Q}(\mathbf{N})\big)\Big)$: a polynomial of degree $\max\{\deg(\mathbf{Q}), \deg(\mathbf{P}) \cdot \deg(\mathbf{Q})\} = \deg(\mathbf{P}) \cdot \deg(\mathbf{Q})$ for non-constant $\mathbf{P}$. This simple observation appears ubiquitously, often implicitly.

ii) An oracle Turing machine $\mathcal{M}^?$ computing $F = F(\varphi, \vec{x})$ is said [13] to run in second-order polynomial time $\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$ if the following holds: On input of any string $\vec{x} \in \{0,1\}^*$ and for any string function oracle $\varphi : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{M}^{\varphi}(\vec{x})$ outputs $F(\varphi, \vec{x})$ and makes at most $\mathbf{\Pi}(|\varphi|, |\vec{x}|)$ steps, where $|\varphi|(m) := \max\big\{|\varphi(\vec{b})| : |\vec{b}| \leq m\big\}$.

iii) Similarly to (i), let us analyze the running time of *oracle* Turing machine composition in (iv)+(v). Indeed, second-order string functionals naturally compose in two distinct ways, similarly to second-order polynomials in Subsection 2.2: For $F, G : (\{0,1\}^*)^{\{0,1\}^*} \times \{0,1\}^* \to \{0,1\}^*$, let $F \circ G \equiv \lambda\varphi.\lambda\vec{x}.\, F\big(\varphi, G(\varphi, \vec{x})\big)$ and $F \star G \equiv \lambda\varphi.\lambda\vec{x}.\, F\big(G(\varphi, \cdot), \vec{x}\big)$, where $G(\varphi, \cdot) \equiv \lambda\vec{y}.G(\varphi, \vec{y})$.

iv) Let $\mathcal{M}^?$ and $\mathcal{N}^?$ denote oracle Turing machines computing F and G, respectively, in second-order running time bounds $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Psi}, \mathbf{M})$ and $\mathbf{\Xi} = \mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})$. Then $F \circ G$ can be computed by an oracle Turing machine in second-order polynomial time $\mathcal{O}\big(\mathbf{\Xi} + \mathbf{\Pi} \circ \mathbf{\Xi}\big)$ of degree $\max\big\{\mathrm{Deg}(\mathbf{\Xi}), \mathrm{Deg}(\mathbf{\Pi}) \cdot \mathrm{Deg}(\mathbf{\Xi})\big\} = \mathrm{Deg}(\mathbf{\Pi}) \cdot \mathrm{Deg}(\mathbf{\Xi})$ for non-constant $\mathbf{\Xi}$.

v) Let $\mathcal{M}^?, \mathcal{N}^?$ and F, G and $\mathbf{\Pi}, \mathbf{\Xi}$ be as in (iv). Then $F \star G$ can be computed by an oracle Turing machine in second-order polynomial time $\mathcal{O}\Big(\big(\mathbf{\Pi} \star \mathbf{\Xi}\big) \cdot \big(\mathbf{\Xi} \circ (\mathbf{\Pi} \star \mathbf{\Xi})\big)\Big)$ of degree $\mathrm{Deg}(\mathbf{\Pi}) \circ \mathrm{Deg}(\mathbf{\Xi}) + \mathrm{Deg}(\mathbf{\Xi}) \cdot \big(\mathrm{Deg}(\mathbf{\Pi}) \circ \mathrm{Deg}(\mathbf{\Xi})\big)$.

vi) [14] designs and analyzes running times of oracle Turing machines computing certain other mixed functionals $H : (\{0,1\}^*)^{\{0,1\}^*} \times \{0,1\}^* \to \{0,1\}^*$. Specifically, [14, Proposition 2.8] establishes a time bound $\big(\mathbf{P} \circ \mathbf{\Phi}\big)^{[r]}(\mathbf{P}(\mathbf{N})) + \mathbf{P}(\mathbf{N})$, where $\mathbf{P} \in \mathbb{N}[N]$ denotes some (unspecified, cf.Table 2) ordinary polynomial and $r \in \mathbb{N}$ indicates how often to iterate $\mathbf{P} \circ \mathbf{\Phi}$. The degree of the this second-order polynomial is $\mathbf{D}^r \cdot \big(\deg(\mathbf{P})\big)^{r+1} = \mathcal{O}(\mathbf{D})^r$. ⌟

The second-order running time bound in Example 14vi) is already rather concise to begin with, hence here considering its second-order degree does not yield as much further simplification as in, say, Example 1. Asymptotic growth w.r.t. $\mathbf{N} \to \infty$ of a second-order polynomial depends additionally on that of its second-order argument $\overline{\mathbf{\Phi}}$, and the second-order degree captures both: cf. Remark 13. In Example 14, the second-order parameter $\overline{\mathbf{\Phi}} = |\varphi| : \mathbb{N}\nearrow\mathbb{N}$ measures the "size" of function-type argument $\varphi : \{0,1\}^* \to \{0,1\}^*$. Other applications, like in Examples 14 above and 17 below, assign other meanings to the second-order parameter:

▶ **Example 15.**

a) To any continuous real function $f : [0;1] \to \mathbb{R}$ (as argument to an operator in Analysis, say) assign as *size* its modulus of uniform continuity [17, Definition 2.12]: $\mu : \mathbb{N}\nearrow\mathbb{N}$ pointwise minimal such that $|x - x'| \leq 2^{-\mu(n)} \Rightarrow |f(x) - f(x')| \leq 2^{-n}$.

b) To any pre-compact metric space (X, d) assign as *size* its Kolmogorov entropy [23]: $\eta : \mathbb{N}\nearrow\mathbb{N}$ pointwise minimal such that there exist $2^{\eta(n)}$ balls of radius 2^{-n} covering X.

c) For $p \geq 1$, to any p-summable sequence $\bar{z}$ assign as *size* its modulus of convergence: $\sigma : \mathbb{N}\nearrow\mathbb{N}$ pointwise minimal such that $2^{-np} \geq \sum_{K \geq 2^{\sigma(n)}} |z_K|^p$.

d) These and more second-order "size" parameters $\overline{\mathbf{\Phi}}$ arise generically from Skolemizing classical $\forall\exists$ statements [4]. We consider moduli as mappings from/to integer exponents w.r.t. base two, see [22, §2.4] and cf. [3, p.186]. ⌟

▶ **Example 16** (1D Real Function Inversion)**.** Recall Example 15a) and record [17, Theorem 2.19] that any polynomial-time computable bijection $f : [0;1] \to [0;1]$ has a polynomial modulus of uniform continuity; and its inverse f^{-1} is again polynomial-time computable – provided that it, too, admits a polynomial modulus of continuity [17, Corollary 4.7].

More generally and more precisely, let $\mathcal{C}_{\mu,\nu}[0;1]$ denote the set of bijections $f : [0;1] \to [0;1]$ having modulus $\leq \mu$ and whose inverse f^{-1} has modulus $\leq \nu$. Note that $f^{-1} \circ f = \mathrm{id}$: the identity on $\mathbb{N}$, with modulus $\mathrm{id} \leq \mu \circ \nu$ and similarly $\nu \circ \mu(n) \geq n$.

i) By the proof of [17, Theorem 4.6], there exists an oracle Turing machine computing the family of inversion operators $\lambda f : \mathcal{C}_{\mu,\nu}[0;1].\ f^{-1} : \mathcal{C}_{\mu,\nu}[0;1]$ in time $\mathcal{O}\Big(\mu\big(\nu(n)\big) + 3\Big)^2$: a second-order polynomial in first-order variable[1] n and in two independent second-order variables μ and ν. This polynomial has degree $2\,\mathbf{E}\,\mathbf{F}$: a *bi*variate first-order polynomial in $\mathbf{E} = \mathrm{Deg}(\mu)$ and $\mathbf{F} = \mathrm{Deg}(\nu)$ according to Remark 13e).

ii) *Tri*section (Appendix A) yields another oracle Turing machine computing the same family of inversion operators in time $\mathcal{O}\Big(n{\cdot}\nu(n{+}3){+}n{\cdot}\mu\big(3{+}\nu(n{+}3)\big)\Big)$: a second-order polynomial of degree $\max\{1+\mathbf{F}, 1+\mathbf{E}\,\mathbf{F}\} \leq \max\{\mathbf{E}\,\mathbf{F}+\mathbf{F}, 2\,\mathbf{E}\,\mathbf{F}\}$ since $\mathbf{E}\,\mathbf{F} = \mathrm{Deg}(\mu \star \nu) \geq 1$ by the above considerations.

iii) Combining μ and ν in the single second-order parameter $\phi := \max\{\mu, \nu\}$, the runtime bound from (i) becomes $\mathcal{O}\Big(\phi\big(\phi(n)\big) + 3\Big)^2$ of univariate degree $2\,\mathbf{D}^2$ in $\mathbf{D} = \mathrm{Deg}(\phi)$; the bound from (ii) becomes $\mathcal{O}\Big(n \cdot \phi\big(3 + \phi(n+3)\big)\Big)$ of univariate degree $1 + \mathbf{D}^2$. ⌟

In Example 16, the degree captures concisely that Trisection (ii) is asymptotically more efficient – and thus preferable over – the algorithm from (i) whenever $\mathbf{E}\,\mathbf{F} > \mathbf{F}$, *i.e.*, when $\mu \circ \nu$ grows faster than ν.

▶ **Example 17** (Continuous Operator in Functional Analysis)**.** Fix $p, q \geq 1$ and recall that $\ell^p = \{\bar{z} : \sum_k |z_k|^p < \infty\}$ denotes the space of p-summable complex sequences $\bar{z} = (z_k)$. Write $\ell^p_R \subseteq \ell^p$ for the sphere with radius $R > 0$, that is, the subset of those sequences having p-norm $\|\bar{z}\|_p \leq R$. For $\sigma : \mathbb{N}\nearrow\mathbb{N}$ let furthermore $\ell^p_{R,\sigma} \subseteq \ell^p_R$ denote the subset of all sequences having modulus of convergence $\leq \sigma$.

We record that $\ell^p_{R,\sigma}$ is compact and, conversely, any compact subset of ℓ^p is contained in $\ell^p_{R,\sigma}$ for some $R \in \mathbb{N}$ and some $\sigma : \mathbb{N}\nearrow\mathbb{N}$.

i) Consider some continuous (but not necessarily linear) $\mathbb{O} : \ell^p_1 \to \ell^q$. By continuity, $\mathbb{O}$ maps compact subsets of ℓ^p to compact subsets of ℓ^q. In particular it maps every $\ell^p_{1,\sigma}$ to $\ell_{R,\tau}$ for some minimal $R = R(\sigma) \in \mathbb{N}$ and some minimal $\tau : \mathbb{N}\nearrow\mathbb{N}$ which also depends on σ. Let $\mathcal{F} = \mathcal{F}_{\mathbb{O}} : (\mathbb{N}\nearrow\mathbb{N}) \to (\mathbb{N}\nearrow\mathbb{N})$ denote the thus well-defined mapping $\lambda\sigma.\tau$ associated with $\mathbb{O}$. Note that $\mathcal{F}$ is non-decreasing: $\sigma' \leq \sigma$ implies $\tau' \leq \tau$. The case $p = 2 = q$ of continuous (linear) operators on Fourier series is particularly relevant [18, Remark 26].

ii) The identity operator $\ell^p \to \ell^p$ has $\mathcal{F} \equiv \lambda\sigma.\sigma$ the identity. The "repetition" operator $\lambda\bar{z} = (z_K)_K : \ell^1.\ (z_{\lfloor K/2 \rfloor})_K : \ell^1$ is linear and 2-Lipschitz and has $\mathcal{F} \equiv \lambda\sigma : (\mathbb{N}\nearrow\mathbb{N}).\ \big(\lambda n : \mathbb{N}.\sigma(n+1)+1\big) : \mathbb{N}$.

iii) For any bijection (aka infinite permutation) $\pi : \mathbb{N} \to \mathbb{N}$, the linear operator $\lambda\bar{z} = (z_K)_K : \ell^p.\ (z_{\pi^{-1}(K)})_K : \ell^p$ is isometric. Its associated $\mathcal{F} = \mathcal{F}_\pi$ is bounded by a second-order polynomial iff $\forall m\, \forall M < 2^m : \pi(M) < 2^{q(m)}$ for some ordinary polynomial q. ⌟

[1] The kind reader may generously forgive us for relaxing the distinction between syntax and semantics.

We close this subsection with some further, sporadic examples:

i) [22, Example 2.23i] involves an expression $\sum_{m=0}^{n+1} \eta(m+1)$, where η denotes the Kolmogorov entropy from Example 15b). This expression in (η, n) itself is not a second-order polynomial, but it is bounded by the second-order polynomial $(n{+}1{+}1){\cdot}\eta(n{+}1{+}1)$ of degree $1 + \mathbf{D}$ with convention $\mathbf{D} = \deg(\eta)$.

ii) [22, Corollary 3.6a] uses the expression $\mathcal{P} \circ \phi \circ \mathcal{P}$, where $\mathcal{P}$ denotes the class of univariate ordinary polynomials. Its degree is $\mathcal{O}(\mathbf{D})$; here the constant hidden in $\mathcal{O}$ captures (the degrees of) the particular ordinary polynomials.

iii) [3, §3] establishes rates of convergence, which turn out to be (bounded by) multivariate second-order polynomials $\mathbf{\Pi}$ in (certain first-order parameters and in) two second-order parameters η and ω; see [3, p.186 ll.8ff]. See also [1]...

2.4 Arctic Second-Order Polynomials

Definition 23d) below considers the degree of a *third*-order polynomial to be an arctic second-order polynomial.

▶ **Definition 18.**

a) *Arctic second-order polynomials in first-order variables* $\vec{\boldsymbol{N}} = (\boldsymbol{N}_1, .. \boldsymbol{N}_M)$ *and one second-order variable* $\mathbf{\Phi}$ *are generated by*

$$\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}} \quad ::= \quad 0 \mid 1 \mid \boldsymbol{N}_1 \mid \ldots \mid \boldsymbol{N}_M \mid \widetilde{\mathbf{\Pi}} + \widetilde{\mathbf{\Xi}} \mid \widetilde{\mathbf{\Pi}} \cdot \widetilde{\mathbf{\Xi}} \mid \mathbf{\Phi}(\widetilde{\mathbf{\Pi}}) \mid \max\big(\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}}\big) \ .$$

with obvious semantics similar to (non-arctic) second-order polynomials.

b) *Regarding syntactic equivalence, extend Rules* (3) *to* max() *capturing monotonicity:*

$$\begin{aligned}
&\max\{\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}}\} \approxeq \max\{\widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Pi}}\}, \qquad \max\big\{\max\{\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}}\}, \widetilde{\mathbf{\Lambda}}\big\} \approxeq \max\big\{\widetilde{\mathbf{\Pi}}, \max\{\widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Lambda}}\}\big\}, \qquad (8)\\
&\widetilde{\mathbf{\Pi}} + \max\{\widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Lambda}}\} \approxeq \max\{\widetilde{\mathbf{\Pi}} + \widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Pi}} + \widetilde{\mathbf{\Lambda}}\}, \quad \widetilde{\mathbf{\Pi}} \cdot \max\{\widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Lambda}}\} \approxeq \max\{\widetilde{\mathbf{\Pi}} \cdot \widetilde{\mathbf{\Xi}}, \widetilde{\mathbf{\Pi}} \cdot \widetilde{\mathbf{\Lambda}}\},\\
&\max\{\widetilde{\mathbf{\Pi}}, 0\} \approxeq \widetilde{\mathbf{\Pi}}, \qquad \big(\widetilde{\mathbf{\Pi}} \approxeq \widetilde{\mathbf{\Pi}}' \ \wedge \ \widetilde{\mathbf{\Xi}} \approxeq \widetilde{\mathbf{\Xi}}'\big) \quad \Rightarrow \quad \max\{\widetilde{\mathbf{\Pi}}, \widetilde{\mathbf{\Xi}}\} \approxeq \max\{\widetilde{\mathbf{\Pi}}', \widetilde{\mathbf{\Xi}}'\}
\end{aligned}$$

Lemma 4 about the asymptotic behaviour of arctic first-order polynomials does not extend to the second-order case:

▶ **Example 19.** Consider $\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N}) := \mathbf{\Phi}^2\big(\mathbf{\Phi}(\mathbf{N})\big) + \mathbf{\Phi}\big(\mathbf{\Phi}(\mathbf{N}^2)\big)$ and $\mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N}) := 2\mathbf{\Phi}\big(\mathbf{\Phi}^2(\mathbf{N})\big)$. They both have the same degree $\mathrm{Deg}\big(\mathbf{\Pi}\big)(\mathbf{D}) = 2\,\mathbf{D}^2 = \mathrm{Deg}\big(\mathbf{\Xi}\big)(\mathbf{D})$. For all integers $a, c, d \geq 2$ and for all sufficiently large $n \in \mathbb{N}$, they evaluate

$$\overline{\mathbf{\Pi}}\big(\lambda m.a \cdot m^d \,,\, n\big) > \overline{\mathbf{\Xi}}\big(\lambda m.a \cdot m^d \,,\, n\big) \ \text{ but } \ \overline{\mathbf{\Pi}}\big(\lambda m.m^d + c \,,\, n\big) < \overline{\mathbf{\Xi}}\big(\lambda m.m^d + c \,,\, n\big).$$

In particular the arctic univariate second-order polynomial $\max\big\{\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N}), \mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})\big\}$ cannot reasonably be said to "asymptotically" coincide with any ordinary second-order polynomial. ⌟

See Appendix B. Due to the semantic completeness Theorem 7, $\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N})$ and $\mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})$ are the only candidates for $\max\big\{\mathbf{\Pi}(\mathbf{\Phi}, \mathbf{N}), \mathbf{\Xi}(\mathbf{\Phi}, \mathbf{N})\big\}$ to "asymptotically" coincide with.

3 Higher-Order Polynomials and Degrees

We now climb further up in the type hierarchy: by one step to the third-order case in Subsection 3.2, and then to the general case in Subsection 3.3. But first revisit and take a new perspective on the first and second-order case:

3.1 First and Second-Order Case Revisited

Section 2 had regarded compositions $\circ$ and $\star$ as operators on previously defined second-order polynomials, and the algebraic properties of these operators were consequences. This is the traditional perspective [13, 15]. In order to proceed to third and higher order polynomials, we now take a new but equivalent perspective on the same families of traditional first and second-order polynomials, considering compositions as part of their syntactic definition:

▶ Remark 20 (First-Order Polynomials, Revisited). Note that the semantics of ordinary polynomials is based on "values", namely starting with 1 and $\overline{\mathbf{N}} \in \mathbb{N}$ and proceeding inductively via $+$ and $\cdot$. Alternatively, ordinary polynomials can be considered as certain mappings

$$\overline{\mathbf{P}} = p \;\equiv\; \lambda\overline{\mathbf{N}} : \mathbb{N}.\, \overline{\mathbf{P}}(\overline{\mathbf{N}}) : \mathbb{N} \;.$$

Here, $+$ and $\cdot$ are tacitly "overloaded" to denote pointwise addition and multiplication of functions instead of values.

a) A univariate first-order polynomial in $\mathbf{N}$ is syntactically generated by the rules

$$\mathbf{P}, \mathbf{Q} \quad ::= \quad 1 \mid \mathbf{N} \mid \mathbf{P} + \mathbf{Q} \mid \mathbf{P} \cdot \mathbf{Q} \mid \mathbf{P} \circ \mathbf{Q} \;\;.$$

b) The semantics of $\mathbf{P}$ is the canonical interpretation as map $\overline{\mathbf{P}} = p : \mathbb{N} \nearrow \mathbb{N}$.

c) Syntactic equivalence is generated by the Rules (3), together with these for $\circ$:

$$1 \circ \mathbf{P} \;\approx\; 1, \qquad \mathbf{N} \circ \mathbf{P} \;\approx\; \mathbf{P} \;\approx\; \mathbf{P} \circ \mathbf{N}, \qquad (\mathbf{P} \circ \mathbf{Q}) \circ \mathbf{R} \;\approx\; \mathbf{P} \circ (\mathbf{Q} \circ \mathbf{R}),$$
$$(\mathbf{P} + \mathbf{Q}) \circ \mathbf{R} \;\approx\; (\mathbf{P} \circ \mathbf{R}) + (\mathbf{Q} \circ \mathbf{R}), \qquad (\mathbf{P} \cdot \mathbf{Q}) \circ \mathbf{R} \;\approx\; (\mathbf{P} \circ \mathbf{R}) \cdot (\mathbf{R} \circ \mathbf{R})$$

d) The degree of $\mathbf{P}$ is defined inductively by

$$\deg(1) = 0, \quad \deg(\mathbf{N}) = 1, \qquad \deg(\mathbf{P} + \mathbf{Q}) = \max\{\deg(\mathbf{P}), \deg(\mathbf{Q})\},$$
$$\deg(\mathbf{P} \cdot \mathbf{Q}) = \deg(\mathbf{P}) + \deg(\mathbf{Q}), \qquad \deg(\mathbf{P} \circ \mathbf{Q}) = \deg(\mathbf{P}) \cdot \deg(\mathbf{Q}).$$

Well-definition follows from (c), together with Commutative/Associative/ Distributive Laws of max() captured in (h).

e) For any first-order polynomial according to (a), there exists a syntactically equivalent first-order polynomial devoid of composition symbol $\circ$.

f) Every classical univariate polynomial (function) can be expressed in the form (a) with semantics (b). Conversely every polynomial according to (a) without composition symbol (e) amounts to a classical polynomial with same semantics.

g) An arctic first-order polynomial in $\mathbf{M}$ is syntactically generated by the rules

$$\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}} \quad ::= \quad 0 \mid 1 \mid \mathbf{M} \mid \widetilde{\mathbf{P}} + \widetilde{\mathbf{Q}} \mid \widetilde{\mathbf{P}} \cdot \widetilde{\mathbf{Q}} \mid \widetilde{\mathbf{P}} \circ \widetilde{\mathbf{Q}} \mid \max\left(\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}}\right) \;\;.$$

h) Arctic syntactic rules extend (c) with: $\max\left\{0, \widetilde{\mathbf{P}}\right\} \approx \widetilde{\mathbf{P}}$, $0 + \widetilde{\mathbf{P}} \approx \widetilde{\mathbf{P}}$, $0 \cdot \widetilde{\mathbf{P}} \approx 0$, $0 \circ \widetilde{\mathbf{P}} \approx 0$,

$$\max\left\{\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}}\right\} \;\approx\; \max\left\{\widetilde{\mathbf{Q}}, \widetilde{\mathbf{P}}\right\}, \quad \max\left\{\widetilde{\mathbf{P}}, \max\left\{\widetilde{\mathbf{Q}}, \widetilde{\mathbf{R}}\right\}\right\} \;\approx\; \max\left\{\max\left\{\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}}\right\}, \widetilde{\mathbf{R}}\right\},$$
$$\widetilde{\mathbf{P}} + \max\left\{\widetilde{\mathbf{Q}}, \widetilde{\mathbf{R}}\right\} \;\approx\; \max\left\{\widetilde{\mathbf{P}} + \widetilde{\mathbf{Q}}, \widetilde{\mathbf{P}} + \widetilde{\mathbf{R}}\right\}, \quad \widetilde{\mathbf{P}} \cdot \max\left\{\widetilde{\mathbf{Q}}, \widetilde{\mathbf{R}}\right\} \;\approx\; \max\left\{\widetilde{\mathbf{P}} \cdot \widetilde{\mathbf{Q}}, \widetilde{\mathbf{P}} \cdot \widetilde{\mathbf{R}}\right\}$$
$$\widetilde{\mathbf{P}} \circ \max\left\{\widetilde{\mathbf{Q}}, \widetilde{\mathbf{R}}\right\} \approx \max\left\{\widetilde{\mathbf{P}} \circ \widetilde{\mathbf{Q}}, \widetilde{\mathbf{P}} \circ \widetilde{\mathbf{R}}\right\},$$
$$\left(\widetilde{\mathbf{P}} \approx \widetilde{\mathbf{P}}' \wedge \widetilde{\mathbf{Q}} \approx \widetilde{\mathbf{Q}}'\right) \Rightarrow \max\left\{\widetilde{\mathbf{P}}, \widetilde{\mathbf{Q}}\right\} \approx \max\left\{\widetilde{\mathbf{P}}', \widetilde{\mathbf{Q}}'\right\}$$ ⌟

Item e) follows from c) by structural induction, which also implies $\mathbf{P} \circ \mathbf{Q} \approx \mathbf{P}' \circ \mathbf{Q}'$.

▶ Remark 21 (Second-Order Polynomials, Revisited). Curry-ing (*Schönfinkel-ing*) suggests considering a second-order polynomial $\mathbf{\Pi}$ as family of first-order polynomials, parameterized by an additional variable $\mathbf{\Phi}$ ranging over $\phi \in (\mathbb{N}\nearrow\mathbb{N})$: $\overline{\mathbf{\Pi}}(\phi,\cdot) \equiv \lambda\overline{\mathbf{N}} : \mathbb{N}.\, \overline{\mathbf{\Pi}}(\phi,\overline{\mathbf{N}}) : \mathbb{N}$. Second-order polynomials can alternatively be interpreted as operators $(\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})$:

$$\overline{\mathbf{\Pi}} \equiv \lambda\phi : (\mathbb{N}\nearrow\mathbb{N}).\, \big(\lambda n : \mathbb{N}.\overline{\mathbf{\Pi}}(\phi,n)\big) : (\mathbb{N}\nearrow\mathbb{N}) \ .$$

Let us re-introduce $\star$ to denote the composition of operators, $\circ$ for composition of functions.

a) A univariate second-order polynomial in $(\mathbf{\Phi},\mathbf{N})$ is syntactically generated by

$$\mathbf{\Pi},\mathbf{\Xi} \quad ::= \quad 1 \mid \mathbf{N} \mid \mathbf{\Phi} \mid \mathbf{\Pi}+\mathbf{\Xi} \mid \mathbf{\Pi}\cdot\mathbf{\Xi} \mid \mathbf{\Pi}\circ\mathbf{\Xi} \mid \mathbf{\Pi}\star\mathbf{\Xi} \ .$$

b) The semantics is given as follows, subject to implicit Curry-ing:

$$\begin{array}{l}
\overline{1} \equiv \lambda\phi : (\mathbb{N}\nearrow\mathbb{N}).\, \lambda n : \mathbb{N}.\ 1 : \mathbb{N}, \quad \overline{\mathbf{N}} \equiv \lambda\phi.\lambda n.\ n : \mathbb{N}, \\
\overline{\mathbf{\Phi}} \equiv \lambda\phi.\ \phi \in (\mathbb{N}\nearrow\mathbb{N}), \qquad \overline{\mathbf{\Pi}+\mathbf{\Xi}} \ \text{ and } \ \overline{\mathbf{\Pi}\cdot\mathbf{\Xi}} \qquad \text{pointwise} \\
\overline{\mathbf{\Pi}\circ\mathbf{\Xi}} \equiv \lambda\phi.\lambda n.\ \overline{\mathbf{\Pi}}\big(\phi,\overline{\mathbf{\Xi}}(\phi,n)\big) : \mathbb{N}, \qquad \overline{\mathbf{\Pi}\star\mathbf{\Xi}} \equiv \lambda\phi : (\mathbb{N}\nearrow\mathbb{N}).\, \overline{\mathbf{\Pi}}\big(\overline{\mathbf{\Xi}}(\phi)\big) : (\mathbb{N}\nearrow\mathbb{N})
\end{array}$$

c) Regarding syntactic equivalence we record, in addition to Remark 20c):

$$\begin{array}{l}
1\star\mathbf{\Pi} \approxeq 1, \quad \mathbf{N}\star\mathbf{\Pi} \approxeq \mathbf{N}, \quad \mathbf{\Phi}\star\mathbf{\Pi} \approxeq \mathbf{\Pi} \approxeq \mathbf{\Pi}\star\mathbf{\Phi}, \quad (\mathbf{\Pi}+\mathbf{\Xi})\star\mathbf{\Lambda} \approxeq (\mathbf{\Pi}\star\mathbf{\Lambda})+(\mathbf{\Xi}\star\mathbf{\Lambda}) \\
(\mathbf{\Pi}\cdot\mathbf{\Xi})\star\mathbf{\Lambda} \approxeq (\mathbf{\Pi}\star\mathbf{\Lambda})\cdot(\mathbf{\Xi}\star\mathbf{\Lambda}), \qquad (\mathbf{\Pi}\circ\mathbf{\Xi})\star\mathbf{\Lambda} \approxeq (\mathbf{\Pi}\star\mathbf{\Lambda})\circ(\mathbf{\Xi}\star\mathbf{\Lambda}) \\
\big(\mathbf{\Pi}\approxeq\mathbf{\Pi}' \ \wedge \ \mathbf{\Xi}\approxeq\mathbf{\Xi}'\big) \quad \Rightarrow \quad \mathbf{\Pi}\circ\mathbf{\Xi} \approxeq \mathbf{\Pi}'\circ\mathbf{\Xi}' \ .
\end{array} \tag{9}$$

d) The degree of $\mathbf{\Pi} = \mathbf{\Pi}(\mathbf{\Phi},\mathbf{N})$ is an arctic first-order polynomial in variable $\mathbf{D} = \mathrm{Deg}(\mathbf{\Phi})$, defined inductively as in Remark 20d) and, additionally, $\mathrm{Deg}(\mathbf{\Pi}\star\mathbf{\Xi}) = \mathrm{Deg}(\mathbf{\Pi})\circ\mathrm{Deg}(\mathbf{\Xi})$.

e) To any 2nd-order polynomial $\mathbf{\Pi}$ according to (a) there is a syntactically equivalent one (i) devoid of composition symbol $\star$ and (ii) whenever $\circ$ occurs, its left argument is $\mathbf{\Phi}$.

f) Any second-order polynomial according to Definition 5 can be expressed in the form (a) with semantics (b). Conversely any polynomial according to (a) satisfying (i)+(ii) from (e) amounts to a 2nd-order polynomial according to Definition 5 with same semantics.

g) An arctic univariate second-order polynomial in $(\mathbf{\Psi},\mathbf{M})$ is syntactically generated by

$$\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}} \quad ::= \quad 0 \mid 1 \mid \mathbf{M} \mid \mathbf{\Psi} \mid \widetilde{\mathbf{\Pi}}+\widetilde{\mathbf{\Xi}} \mid \widetilde{\mathbf{\Pi}}\cdot\widetilde{\mathbf{\Xi}} \mid \widetilde{\mathbf{\Pi}}\circ\widetilde{\mathbf{\Xi}} \mid \widetilde{\mathbf{\Pi}}\star\widetilde{\mathbf{\Xi}} \mid \max\big(\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}}\big) \ .$$

h) Arctic syntactic rules extend (c), Remark 20h), and (8) in Definition 18b) with: $\widetilde{\mathbf{\Pi}}\star\max\big\{\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Lambda}}\big\} \approxeq \max\big\{\widetilde{\mathbf{\Pi}}\star\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Pi}}\star\widetilde{\mathbf{\Lambda}}\big\}$. ⌟

Items e+f) record that this new perspective coincides with Definition 5. Note that now both $\mathbf{\Phi}$ and $\mathbf{N}$ are of type $(\mathbb{N}\nearrow\mathbb{N})$ but, other than for composition "$\star$", the type of its two arguments tacitly gets Curry-ed for composition "$\circ$"; and Proposition 12 is now axiomatized in Item d). Again, Item e) follows from structural induction using Item c), and implies congruence w.r.t. $\star$; but congruence w.r.t. $\circ$ now needs to be postulated as Rule (9). Regarding the first part of Item f), rewrite $\mathbf{\Phi}(\mathbf{\Pi})$ as $\mathbf{\Phi}\circ\mathbf{\Pi}$. Second-order polynomials according to Remark 21 use brackets only to express priority, not anymore to express "application".

3.2 Third-Order Polynomials and Degrees

Second-order polynomials describe "tame" dependencies on both integer arguments $n \in \mathbb{N}$ and integer function arguments $\phi : \mathbb{N} \nearrow \mathbb{N}$. Third-order polynomials should additionally take into account dependency on integer operator arguments $\mathcal{F} : (\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})$. Recall Example 17 where continuous operators between Banach spaces give rise to all possible such $\mathcal{F}$. In addition to variable $\mathbf{N}$ ranging over $\mathbb{N}$ and $\mathbf{\Phi}$ ranging over $(\mathbb{N}\nearrow\mathbb{N})$, now introduce indeterminate $\mathfrak{F}$ to range over the set $\big((\mathbb{N}\nearrow\mathbb{N}) \nearrow (\mathbb{N}\nearrow\mathbb{N})\big)$ of total monotonic integer operators.

▶ Remark 22. Curry-ing respects monotonicity: $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is non-decreasing iff $f : \mathbb{N} \to (\mathbb{N} \nearrow \mathbb{N})$ is non-decreasing and well-defined regarding its co-domain. Write $f : \mathbb{N} \nearrow \mathbb{N} \nearrow \mathbb{N}$.

A third-order polynomial can thus be considered as a certain family of monotonic first-order polynomials, parameterized monotonically by two variables $\boldsymbol{\Phi}$ and $\boldsymbol{\mathfrak{F}}$ that range over second-order and third-order arguments $\phi \in (\mathbb{N} \nearrow \mathbb{N})$ and $\mathcal{F} \in \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big)$, respectively; or alternatively as a family of monotonic second-order polynomials, parameterized monotonically by $\mathcal{F}$; or as monotonic "hyper"-operator of type $\big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big) \nearrow \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big)$. We introduce $\#$ to denote the composition of such hyper-operators.

After first and second-order Remarks 20 and 21, the following Definition 23 and Theorem 25 about third-order polynomials and their degrees follow naturally:

▶ **Definition 23.**

a) *A* third-*order polynomial in* $(\boldsymbol{\mathfrak{F}}, \boldsymbol{\Phi}, \boldsymbol{N})$ *is syntactically generated by*

$$\boldsymbol{\mathfrak{P}}, \boldsymbol{\mathfrak{Q}} \quad ::= \quad 1 \mid \boldsymbol{N} \mid \boldsymbol{\Phi} \mid \boldsymbol{\mathfrak{F}} \mid \boldsymbol{\mathfrak{P}} + \boldsymbol{\mathfrak{Q}} \mid \boldsymbol{\mathfrak{P}} \cdot \boldsymbol{\mathfrak{Q}} \mid \boldsymbol{\mathfrak{P}} \circ \boldsymbol{\mathfrak{Q}} \mid \boldsymbol{\mathfrak{P}} \star \boldsymbol{\mathfrak{Q}} \mid \boldsymbol{\mathfrak{P}} \# \boldsymbol{\mathfrak{Q}}.$$

b) *Its semantics is* $\overline{\boldsymbol{\mathfrak{P}} \# \boldsymbol{\mathfrak{Q}}} \equiv \lambda \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big).\, \overline{\boldsymbol{\mathfrak{P}}}\big(\overline{\boldsymbol{\mathfrak{Q}}}(\mathcal{F})\big) : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big)$,

$$\begin{aligned}
\overline{1} &\equiv \lambda \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big). \lambda \phi : (\mathbb{N} \nearrow \mathbb{N}). \lambda n : \mathbb{N}.\ 1 : \mathbb{N} \\
\overline{\boldsymbol{N}} &\equiv \lambda \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big). \lambda \phi : (\mathbb{N} \nearrow \mathbb{N}). \lambda n.\ n : \mathbb{N} \\
\overline{\boldsymbol{\Phi}} &\equiv \lambda \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big). \lambda \phi : (\mathbb{N} \nearrow \mathbb{N}).\ \phi : (\mathbb{N} \nearrow \mathbb{N}) \\
\overline{\boldsymbol{\mathfrak{F}}} &\equiv \lambda \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big).\ \mathcal{F} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big)
\end{aligned}$$

all of (or Curry-ed to) type $\big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big) \nearrow \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big)$. *Moreover*

$$\begin{aligned}
\overline{\boldsymbol{\mathfrak{P}} + \boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n) &= \overline{\boldsymbol{\mathfrak{P}}}(\mathcal{F}, \phi, n) + \overline{\boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n) \\
\overline{\boldsymbol{\mathfrak{P}} \cdot \boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n) &= \overline{\boldsymbol{\mathfrak{P}}}(\mathcal{F}, \phi, n) \cdot \overline{\boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n) \\
\overline{\boldsymbol{\mathfrak{P}} \circ \boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n) &= \overline{\boldsymbol{\mathfrak{P}}}\big(\mathcal{F}, \phi, \overline{\boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi, n)\big) : \mathbb{N} \\
\overline{\boldsymbol{\mathfrak{P}} \star \boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi) &= \overline{\boldsymbol{\mathfrak{P}}}(\mathcal{F}, \overline{\boldsymbol{\mathfrak{Q}}}(\mathcal{F}, \phi)) : (\mathbb{N} \nearrow \mathbb{N})
\end{aligned}$$

c) *In addition to Remark 20c) and Remark 21c), we have syntactic equivalence rules*

$$\begin{aligned}
&1\#\boldsymbol{\mathfrak{P}} \approx 1, \qquad \boldsymbol{N}\#\boldsymbol{\mathfrak{P}} \approx \boldsymbol{N}, \qquad \boldsymbol{\Phi}\#\boldsymbol{\mathfrak{P}} \approx \boldsymbol{\Phi}, \qquad \boldsymbol{\mathfrak{F}}\#\boldsymbol{\mathfrak{P}} \approx \boldsymbol{\mathfrak{P}} \approx \boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{F}} \\
&(\boldsymbol{\mathfrak{P}} \star \boldsymbol{\mathfrak{Q}})\#\boldsymbol{\mathfrak{R}} \approx (\boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{R}}) \star (\boldsymbol{\mathfrak{Q}}\#\boldsymbol{\mathfrak{R}}), \qquad (\boldsymbol{\mathfrak{P}} \circ \boldsymbol{\mathfrak{Q}})\#\boldsymbol{\mathfrak{R}} \approx (\boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{R}}) \circ (\boldsymbol{\mathfrak{Q}}\#\boldsymbol{\mathfrak{R}}), \\
&(\boldsymbol{\mathfrak{P}} + \boldsymbol{\mathfrak{Q}})\#\boldsymbol{\mathfrak{R}} \approx (\boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{R}}) + (\boldsymbol{\mathfrak{Q}}\#\boldsymbol{\mathfrak{R}}), \qquad (\boldsymbol{\mathfrak{P}} \cdot \boldsymbol{\mathfrak{Q}})\#\boldsymbol{\mathfrak{R}} \approx (\boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{R}}) \cdot (\boldsymbol{\mathfrak{Q}}\#\boldsymbol{\mathfrak{R}}) \\
&\big(\boldsymbol{\mathfrak{P}} \approx \boldsymbol{\mathfrak{P}}' \ \wedge \ \boldsymbol{\mathfrak{Q}} \approx \boldsymbol{\mathfrak{Q}}'\big) \quad \Rightarrow \quad \boldsymbol{\mathfrak{P}} \star \boldsymbol{\mathfrak{Q}} \approx \boldsymbol{\mathfrak{P}}' \star \boldsymbol{\mathfrak{Q}}' \ .
\end{aligned}$$

d) *The* degree *of third-order polynomial* $\boldsymbol{\mathfrak{P}}$ *is an arctic second-order polynomial in first-order variable* $\boldsymbol{D} = \mathrm{Deg}(\boldsymbol{\Phi})$ *and second-order variable* $\boldsymbol{\Psi} = \mathrm{DEG}(\mathcal{F})$. *It is defined inductively as in Remark 20d) and Remark 21d) and, additionally,* $\mathrm{DEG}(\boldsymbol{\mathfrak{P}}\#\boldsymbol{\mathfrak{Q}}) := \mathrm{DEG}(\boldsymbol{\mathfrak{P}}) \star \mathrm{DEG}(\boldsymbol{\mathfrak{Q}})$.

e) *An* arctic *third-order polynomial in* $(\boldsymbol{\mathfrak{G}}, \boldsymbol{\Psi}, \boldsymbol{M})$ *is syntactically generated by*

$$\widetilde{\boldsymbol{\mathfrak{P}}}, \widetilde{\boldsymbol{\mathfrak{Q}}} \quad ::= \quad 0 \mid 1 \mid \boldsymbol{M} \mid \boldsymbol{\Psi} \mid \boldsymbol{\mathfrak{G}} \mid \widetilde{\boldsymbol{\mathfrak{P}}} + \widetilde{\boldsymbol{\mathfrak{Q}}} \mid \widetilde{\boldsymbol{\mathfrak{P}}} \cdot \widetilde{\boldsymbol{\mathfrak{Q}}} \mid \widetilde{\boldsymbol{\mathfrak{P}}} \circ \widetilde{\boldsymbol{\mathfrak{Q}}} \mid \widetilde{\boldsymbol{\mathfrak{P}}} \star \widetilde{\boldsymbol{\mathfrak{Q}}} \mid \widetilde{\boldsymbol{\mathfrak{P}}} \# \widetilde{\boldsymbol{\mathfrak{Q}}} \mid \max\big(\widetilde{\boldsymbol{\mathfrak{P}}}, \widetilde{\boldsymbol{\mathfrak{Q}}}\big).$$

f) *Syntactic rules extend (c) and Remark 21h) with* $\widetilde{\boldsymbol{\Pi}} \# \max\big\{\widetilde{\boldsymbol{\Xi}}, \widetilde{\boldsymbol{\Lambda}}\big\} \approx \max\big\{\widetilde{\boldsymbol{\Pi}}\#\widetilde{\boldsymbol{\Xi}}, \widetilde{\boldsymbol{\Pi}}\#\widetilde{\boldsymbol{\Lambda}}\big\}$.

▶ **Example 24.**

a) Consider the following third-order polynomial

$$\mathfrak{P}(\mathfrak{F},\mathbf{\Phi},\mathbf{N}) \;:=\; \big(\mathfrak{F}\star\mathfrak{F}^4\star\big(\mathbf{\Phi}^2\circ\mathbf{\Phi}\circ\big(2\cdot\mathbf{N}^6+7\big)\big)\big) \;+\; \big(\mathfrak{F}^3\star\big(\mathbf{\Phi}\circ\mathbf{\Phi}^3\circ\mathbf{\Phi}^5\circ\mathbf{N}^4\big)\big)\;,$$

where "$\mathfrak{O}^3$" abbreviates $(\mathfrak{O}\cdot\mathfrak{O}\cdot\mathfrak{O})$. $\mathfrak{P}$ has degree

$$\mathrm{DEG}\,(\mathfrak{P})(\mathbf{\Psi},\mathbf{D}) \;=\; \max\big\{\mathbf{\Psi}\big(4\mathbf{\Psi}(12\,\mathbf{D}^2)\big)\,,\;3\mathbf{\Psi}(60\,\mathbf{D}^3)\big\}$$

and semantics $\overline{\mathfrak{P}} \;\equiv\; \lambda\mathcal{F}.\,\lambda\phi.\;\mathcal{F}\big(\phi^4\big(\lambda n.\phi^2(\phi(2n^6+7))\big)\big)\;+\;\mathcal{F}^3\big(\lambda n.\phi\big(\phi^3(\phi^5(n^4))\big)\big)$

b) By Remark 30g) in Subsection 3.4 below, the following mapping of type $\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)\nearrow\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)$ is *not* (the semantics of) a third-order polynomial:

$$\lambda\mathcal{F}:\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big).\;\lambda\phi:(\mathbb{N}\nearrow\mathbb{N}).\;\lambda n:\mathbb{N}.\quad\mathcal{F}\big(\lambda m:\mathbb{N}\,.4m^3+5\big)(n^2+n+1):\mathbb{N}$$

c) A first-order polynomial is a second-order polynomial $\mathbf{\Pi}(\mathbf{\Phi},\mathbf{N})$ which does not "depend" on $\mathbf{\Phi}$, *i.e.*, whose semantics of type $(\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})$ is a constant of type $(\mathbb{N}\nearrow\mathbb{N})$; recall Theorem 7. A second-order polynomial is a third-order polynomial whose semantics of type $\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)\nearrow\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)$ is a constant of type $\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)$. ⌟

▶ **Theorem 25.**

a) *Every third-order polynomial as in Definition 23a) can be syntactically transformed, using Definition 23c), into an equivalent form in which (i) # does not occur; (ii) whenever $\star$ occurs, its left argument is $\mathfrak{F}$; (iii) whenever $\circ$ occurs, its left argument is either $\mathfrak{F}$ or $\mathbf{\Phi}$ or of the form $(\mathfrak{P}\star\mathfrak{O})$.*

b) *The degree of third-order polynomials is well-defined, namely invariant under the syntactic congruence relations from Definition 23c)+e).*

▶ **Example 26.** Consider the third-order polynomial $\mathfrak{P}+\mathfrak{O}$ in $(\mathfrak{F},\mathbf{\Psi},\mathbf{D})$

$$\mathfrak{P}\;+\;\mathfrak{O}\;\;=\;\;\mathfrak{F}\star\mathfrak{F}\star(\mathbf{\Psi}\circ\mathbf{\Psi})\;+\;\mathfrak{F}\star(\mathfrak{F}\circ\mathfrak{F})\;\cdot\;(\mathfrak{F}\star\mathfrak{F})\circ(\mathfrak{F}\star\mathfrak{F})$$

with implicit default operator precedence + after · after ∘ after $\star$. Noting that $\mathfrak{F}(\mathbf{\Psi})=\mathfrak{F}$, the semantics $\overline{\mathfrak{P}+\mathfrak{O}}$ of $\mathfrak{P}+\mathfrak{O}$ means mapping argument pairs $\psi:(\mathbb{N}\nearrow\mathbb{N})$ and $\mathcal{F}:\big((\mathbb{N}\nearrow\mathbb{N})\nearrow(\mathbb{N}\nearrow\mathbb{N})\big)$ to the value $\mathcal{F}\big(\mathcal{F}(\psi\circ\psi)\big)\;+\;\mathcal{F}\big(\mathcal{F}(\psi)\circ\mathcal{F}(\psi)\big)\;\cdot\;\mathcal{F}\big(\mathcal{F}(\psi)\big)\circ\mathcal{F}\big(\mathcal{F}(\psi)\big)$. $\mathrm{DEG}(\mathfrak{P}+\mathfrak{O})$ here coincides with the arctic second-order polynomial from Example 19.

3.3 Polynomials and Degrees of Arbitrary Order

Motivated by [11] and [10, §3.3.1], we are now ready to treat the general case:

▶ **Definition 27.** *Let $\delta\in\mathbb{N}$ denote the order of the polynomials under consideration.*

a) *With "$\mathbb{N}\overset{\delta}{\nearrow}\mathbb{N}$" we abbreviate the type $\big(\mathbb{N}\overset{\delta-1}{\nearrow}\mathbb{N}\big)\nearrow\big(\mathbb{N}\overset{\delta-1}{\nearrow}\mathbb{N}\big)$ in case $\delta\geq 1$; "$\mathbb{N}\overset{0}{\nearrow}\mathbb{N}$" means $\mathbb{N}$, and "$\mathbb{N}\overset{-1}{\nearrow}\mathbb{N}$" is $\{1\}\subset\mathbb{N}$. Let "$\overset{0}{\star}$" mean multiplication "$\cdot$" and $\overset{1}{\star}=+$" addition and, for $\delta\geq 1$, "$\overset{\delta}{\star}$" denotes the typed composition $g\circ f$ of $f,g:\big(\mathbb{N}\overset{\delta}{\nearrow}\mathbb{N}\big)$.*
In the sequel, variable $\boldsymbol{V}^{(\delta)}$ ranges over values $\overline{\boldsymbol{V}}^{(\delta)}=V^{(\delta)}:\mathbb{N}\overset{\delta-1}{\nearrow}\mathbb{N}$; $\boldsymbol{V}^{(0)}=1$.

b) *Univariate* order-δ polynomials *in variables $(\boldsymbol{V}^{(\delta)},\ldots,\boldsymbol{V}^{(1)})$ are syntactically generated by*

$$\mathbf{\Pi},\mathbf{\Xi}\quad ::=\quad 1\mid\boldsymbol{V}^{(1)}\mid\cdots\mid\boldsymbol{V}^{(\delta)}\mid\mathbf{\Pi}+\mathbf{\Xi}\mid\mathbf{\Pi}\cdot\mathbf{\Xi}\mid\mathbf{\Pi}\overset{1}{\star}\mathbf{\Xi}\mid\cdots\mid\mathbf{\Pi}\overset{\delta}{\star}\mathbf{\Xi}\;.$$

c) *The semantics is $\overline{\mathbf{\Pi}}\;:\;\mathbb{N}\overset{\delta}{\nearrow}\mathbb{N}\;=\;\big(\mathbb{N}\overset{\delta-1}{\nearrow}\mathbb{N}\big)\nearrow\big(\mathbb{N}\overset{\delta-1}{\nearrow}\mathbb{N}\big)$, where*

$$\overline{\boldsymbol{V}}^{(\delta)}\;\equiv\;\big(\lambda V^{(\delta)}.\,V^{(\delta)}\big)\quad\cdots\qquad\qquad\overline{\boldsymbol{V}}^{(2)}\;\equiv\;\big(\lambda V^{(\delta)}\ldots\lambda V^{(2)}.\,V^{(2)}\big)$$
$$\overline{\boldsymbol{V}}^{(1)}\;\equiv\;\big(\lambda V^{(\delta)}\ldots\lambda V^{(1)}.\,V^{(1)}\big)\qquad\qquad\overline{1}\;\equiv\;\big(\lambda V^{(\delta)}\ldots\lambda V^{(1)}.\,1\big)$$

are all understood via Curry-ing as of type $\big(\mathbb{N}\overset{\delta}{\nearrow}\mathbb{N}\big)$. Moreover, as structural induction:

$$
\begin{aligned}
\overline{\mathbf{\Pi}+\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(1)}) &= \overline{\mathbf{\Pi}}(V^{(\delta)},\dots,V^{(1)})+\overline{\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(1)}) \;:\mathbb{N}\\
\overline{\mathbf{\Pi}\cdot\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(1)}) &= \overline{\mathbf{\Pi}}(V^{(\delta)},\dots,V^{(1)})\cdot\overline{\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(1)}) \;:\mathbb{N}\\
\overline{\mathbf{\Pi}\,{}^{1}\!\!\star\,\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(1)}) &= \overline{\mathbf{\Pi}}\Big(V^{(\delta)},\dots V^{(2)},\overline{\mathbf{\Xi}}(V^{(\delta)},\dots V^{(1)})\Big) \;:\mathbb{N}=\mathbb{N}\,{}^{0}\!\nearrow\mathbb{N}\\
\overline{\mathbf{\Pi}\,{}^{2}\!\!\star\,\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(2)}) &= \overline{\mathbf{\Pi}}\Big(V^{(\delta)},\dots,V^{(3)},\overline{\mathbf{\Xi}}(V^{(\delta)},\dots,V^{(2)})\Big) \;:(\mathbb{N}\,{}^{1}\!\nearrow\mathbb{N})\\
&\vdots\\
\overline{\mathbf{\Pi}\,{}^{\delta-1}\!\!\star\,\mathbf{\Xi}}(V^{(\delta)},V^{(\delta-1)}) &= \overline{\mathbf{\Pi}}\Big(V^{(\delta)},\overline{\mathbf{\Xi}}(V^{(\delta)},V^{(\delta-1)})\Big) \;:(\mathbb{N}\,{}^{\delta-2}\!\nearrow\mathbb{N})\\
\overline{\mathbf{\Pi}\,{}^{\delta}\!\!\star\,\mathbf{\Xi}}(V^{(\delta)}) &= \overline{\mathbf{\Pi}}\Big(\overline{\mathbf{\Xi}}(V^{(\delta)})\Big) \;:(\mathbb{N}\,{}^{\delta-1}\!\nearrow\mathbb{N})
\end{aligned}
$$

d) *Syntactic rules are:* $\mathbf{\Pi}+\mathbf{\Xi}\approx\mathbf{\Xi}+\mathbf{\Pi}$, $\mathbf{\Pi}\cdot\mathbf{\Xi}\approx\mathbf{\Xi}\cdot\mathbf{\Pi}$,

$$
\begin{aligned}
&1\cdot\mathbf{\Pi}\approx\mathbf{\Pi}, \qquad 1\,{}^{1}\!\!\star\,\mathbf{\Pi}\approx 1, \qquad \dots \qquad 1\,{}^{\delta}\!\!\star\,\mathbf{\Pi}\approx 1\\
&\boldsymbol{V}^{(1)}\,{}^{1}\!\!\star\,\mathbf{\Pi}\approx\mathbf{\Pi}, \quad \boldsymbol{V}^{(1)}\,{}^{2}\!\!\star\,\mathbf{\Pi}\approx\boldsymbol{V}^{(1)}, \quad \dots \quad \boldsymbol{V}^{(1)}\,{}^{\delta}\!\!\star\,\mathbf{\Pi}\approx\boldsymbol{V}^{(1)}\\
&\quad\vdots\\
&\boldsymbol{V}^{(\delta-1)}\,{}^{\delta-1}\!\!\star\,\mathbf{\Pi}\approx\mathbf{\Pi}, \quad \boldsymbol{V}^{(\delta-1)}\,{}^{\delta}\!\!\star\,\mathbf{\Pi}\approx\boldsymbol{V}^{(\delta-1)}, \qquad \boldsymbol{V}^{(\delta)}\,{}^{\delta}\!\!\star\,\mathbf{\Pi}\approx\mathbf{\Pi}
\end{aligned}
$$

$$
\begin{aligned}
&(\mathbf{\Pi}+\mathbf{\Xi})+\mathbf{\Lambda}\approx\mathbf{\Pi}+(\mathbf{\Xi}+\mathbf{\Lambda}), \quad (\mathbf{\Pi}+\mathbf{\Xi})\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}\approx(\mathbf{\Pi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda})+(\mathbf{\Xi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}) \quad (\varepsilon=0\dots\delta)\\
&(\mathbf{\Pi}\cdot\mathbf{\Xi})\cdot\mathbf{\Lambda}\approx\mathbf{\Pi}\cdot(\mathbf{\Xi}\cdot\mathbf{\Lambda}), \qquad (\mathbf{\Pi}\cdot\mathbf{\Xi})\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}\approx(\mathbf{\Pi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda})\cdot(\mathbf{\Xi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}) \quad (\varepsilon=1\dots\delta)\\
&(\mathbf{\Pi}\,{}^{1}\!\!\star\,\mathbf{\Xi})\,{}^{1}\!\!\star\,\mathbf{\Lambda}\approx\mathbf{\Pi}\,{}^{1}\!\!\star\,(\mathbf{\Xi}\,{}^{1}\!\!\star\,\mathbf{\Lambda}), \qquad (\mathbf{\Pi}\,{}^{1}\!\!\star\,\mathbf{\Xi})\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}\approx(\mathbf{\Pi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda})\,{}^{1}\!\!\star\,(\mathbf{\Xi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Lambda}) \quad (\varepsilon=2\dots\delta)\\
&\quad\vdots\\
&(\mathbf{\Pi}\,{}^{\delta-1}\!\!\star\,\mathbf{\Xi})\,{}^{\delta-1}\!\!\star\,\mathbf{\Lambda}\approx\mathbf{\Pi}\,{}^{\delta-1}\!\!\star\,(\mathbf{\Lambda}\,{}^{\delta-1}\!\!\star\,\mathbf{\Xi}), \qquad (\mathbf{\Pi}\,{}^{\delta-1}\!\!\star\,\mathbf{\Xi})\,{}^{\delta}\!\!\star\,\mathbf{\Lambda}\approx(\mathbf{\Pi}\,{}^{\delta}\!\!\star\,\mathbf{\Lambda})\,{}^{\delta-1}\!\!\star\,(\mathbf{\Xi}\,{}^{\delta}\!\!\star\,\mathbf{\Lambda})\\
&(\mathbf{\Pi}\approx\mathbf{\Pi}'\ \wedge\ \mathbf{\Xi}\approx\mathbf{\Xi}')\ \Rightarrow\ \mathbf{\Pi}\,{}^{\varepsilon}\!\!\star\,\mathbf{\Xi}\approx\mathbf{\Pi}'\,{}^{\varepsilon}\!\!\star\,\mathbf{\Xi}' \quad (\varepsilon=-1,0\dots\delta-1)\ .
\end{aligned}
$$

e) *A univariate* arctic *order-δ polynomial in* $(\boldsymbol{W}^{(\delta)},\dots,\boldsymbol{W}^{(1)})$ *is syntactically generated by*

$$
\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}} \quad ::= \quad 0\mid 1\mid \boldsymbol{W}^{(1)}\mid\cdots\mid\boldsymbol{W}^{(\delta)}\mid\widetilde{\mathbf{\Pi}}+\widetilde{\mathbf{\Xi}}\mid\widetilde{\mathbf{\Pi}}\cdot\widetilde{\mathbf{\Xi}}\mid\widetilde{\mathbf{\Pi}}\,{}^{1}\!\!\star\,\widetilde{\mathbf{\Xi}}\mid\cdots\mid\widetilde{\mathbf{\Pi}}\,{}^{\delta}\!\!\star\,\widetilde{\mathbf{\Xi}}\mid\max\big(\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}}\big)\ .
$$

f) *Syntactic equivalence among* arctic *order-δ polynomials extends (d) with these rules:*

$$
\begin{aligned}
&\max\big\{0,\widetilde{\mathbf{\Pi}}\big\}\approx\widetilde{\mathbf{\Pi}}, \quad 0+\widetilde{\mathbf{\Pi}}\approx\widetilde{\mathbf{\Pi}}, \qquad 0\,{}^{\varepsilon}\!\!\star\,\widetilde{\mathbf{\Pi}}\approx 0 \quad (\varepsilon=0\dots\delta)\\
&\max\big\{\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}}\big\}\approx\max\big\{\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Pi}}\big\}, \quad \max\Big\{\widetilde{\mathbf{\Pi}},\max\big\{\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Lambda}}\big\}\Big\}\approx\max\Big\{\max\big\{\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}}\big\},\widetilde{\mathbf{\Lambda}}\Big\},\\
&\widetilde{\mathbf{\Pi}}\,{}^{\varepsilon}\!\!\star\,\max\big\{\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Lambda}}\big\}\approx\max\big\{\widetilde{\mathbf{\Pi}}\,{}^{\varepsilon}\!\!\star\,\widetilde{\mathbf{\Xi}},\widetilde{\mathbf{\Pi}}\,{}^{\varepsilon}\!\!\star\,\widetilde{\mathbf{\Lambda}}\big\} \quad (\varepsilon=-1,0\dots\delta)\\
&(\widetilde{\mathbf{\Pi}}\approx\widetilde{\mathbf{\Pi}}'\ \wedge\ \widetilde{\mathbf{\Xi}}\approx\widetilde{\mathbf{\Xi}}')\ \Rightarrow\ \max\big\{\widetilde{\mathbf{\Pi}},\widetilde{\mathbf{\Xi}}\big\}\approx\max\big\{\widetilde{\mathbf{\Pi}}',\widetilde{\mathbf{\Xi}}'\big\}\ .
\end{aligned}
$$

g) *The* degree $\mathrm{Deg}(\mathbf{\Pi})$ *of (non-arctic) order-δ polynomial* $\mathbf{\Pi}=\mathbf{\Pi}(\boldsymbol{V}^{(\delta)},\dots,\boldsymbol{V}^{(1)})$ *is the arctic order-$(\delta-1)$ polynomial in* $(\boldsymbol{W}^{(\delta-1)},\dots,\boldsymbol{W}^{(1)})$ *defined inductively by*

$$
\begin{aligned}
&\mathrm{Deg}(1)=0, \quad \mathrm{Deg}(\boldsymbol{V}^{(1)})=1, \quad \mathrm{Deg}(\boldsymbol{V}^{(2)})=\boldsymbol{W}^{(1)},\ \dots\ \mathrm{Deg}(\boldsymbol{V}^{(\delta)})=\boldsymbol{W}^{(\delta-1)}\\
&\mathrm{Deg}(\mathbf{\Pi}+\mathbf{\Xi})=\max\big\{\mathrm{Deg}(\mathbf{\Pi}),\mathrm{Deg}(\mathbf{\Xi})\big\}, \qquad \mathrm{Deg}(\mathbf{\Pi}\cdot\mathbf{\Xi})=\mathrm{Deg}(\mathbf{\Pi})+\mathrm{Deg}(\mathbf{\Xi}),\\
&\mathrm{Deg}(\mathbf{\Pi}\,{}^{1}\!\!\star\,\mathbf{\Xi})=\mathrm{Deg}(\mathbf{\Pi})\cdot\mathrm{Deg}(\mathbf{\Xi}), \quad \dots \quad \mathrm{Deg}(\mathbf{\Pi}\,{}^{\delta}\!\!\star\,\mathbf{\Xi})=\mathrm{Deg}(\mathbf{\Pi})\,{}^{\delta-1}\!\!\star\,\mathrm{Deg}(\mathbf{\Xi})\ .
\end{aligned}
$$

▶ **Theorem 28.**

a) *Every δ-order polynomial as in Definition 27b) can be syntactically transformed, using Definition 27d), into an equivalent form in which (i) $\overset{\delta}{\star}$ does not occur; (ii) whenever $\overset{\delta-1}{\star}$ occurs, its left argument is $\boldsymbol{V}^{(\delta)}$; (iii) whenever $\overset{\delta-2}{\star}$ occurs, its left argument is $\boldsymbol{V}^{(\delta)}$ or $\boldsymbol{V}^{(\delta-1)}$ or of the form $(\boldsymbol{\Pi} \overset{\delta-1}{\star} \boldsymbol{\Xi})$; (iv) whenever $\overset{\delta-3}{\star}$ occurs, its left argument is $\boldsymbol{V}^{(\delta)}$ or $\boldsymbol{V}^{(\delta-1)}$ or $\boldsymbol{V}^{(\delta-2)}$ or of the form $(\boldsymbol{\Pi} \overset{\delta-1}{\star} \boldsymbol{\Xi})$ or of the form $(\boldsymbol{\Pi} \overset{\delta-2}{\star} \boldsymbol{\Xi})$.*

b) *The degree of δ-order polynomials is well-defined, namely invariant under syntactic equivalences according to Definition 27d+f).*

Proofs proceed as in the third-order case. We hope that Remark 2e) and Theorem 7 generalize:

▶ **Conjecture 29.** *Let $\boldsymbol{\Pi} \approx\!\!\!\!\!\!\approx \boldsymbol{\Xi}$ denote two syntactically non-equivalent order-δ polynomials. Then there exists an assignment $\overline{\boldsymbol{V}}^{(1)} = V^{(1)} : \mathbb{N} = \big(\mathbb{N} \overset{0}{\nearrow} \mathbb{N}\big)$, $\overline{\boldsymbol{V}}^{(2)} = V^{(2)} : \big(\mathbb{N} \overset{1}{\nearrow} \mathbb{N}\big)$, $\ldots\, \overline{\boldsymbol{V}}^{(\delta)} = V^{(\delta)} : \big(\mathbb{N} \overset{\delta-1}{\nearrow} \mathbb{N}\big)$ such that $\overline{\boldsymbol{\Pi}}\big(V^{(\delta)}, \ldots, V^{(1)}\big) \neq \overline{\boldsymbol{\Xi}}\big(V^{(\delta)}, \ldots, V^{(1)}\big) \in \mathbb{N}$.*

3.4 Simply Typed Lambda Calculus Perspective

We have already employed Lambda notation, for instance when expressing the semantics of higher-order polynomials in Remark 21b) and in Definitions 23b) and 27c). A reader familiar with Type Theory might appreciate the following alternative characterization of order-δ polynomials, here for simplicity only in the non-arctic case.

▶ Remark 30.

a) Let $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\delta)}$ denote variables, and $\mathbf{V}^{(0)}$ a constant symbol. For each $\varepsilon = -1, 0, 1, \ldots, \delta$ let $\overset{\varepsilon}{\star}$ denote an associative binary function symbol.

b) Write $\mathcal{P}^{(\delta)}$ for the set of syntactically valid expressions $\boldsymbol{\Pi}^{(\delta)}$ over said variables and symbols. Order-δ polynomials are Lambda Terms of the form

$$\lambda\, \mathbf{V}^{(1)} .\ \ldots\ \lambda\, \mathbf{V}^{(\delta)} .\ \boldsymbol{\Pi}^{(\delta)} \ . \tag{10}$$

c) Towards the semantics of such a Lambda Term, impose initial types as follows:

$$\mathbf{V}^{(0)} = 1 : \{1\} = (\mathbb{N} \overset{-1}{\nearrow} \mathbb{N}), \quad \mathbf{V}^{(1)} : \mathbb{N} = (\mathbb{N} \overset{0}{\nearrow} \mathbb{N}), \quad \mathbf{V}^{(2)} : (\mathbb{N} \nearrow \mathbb{N}),$$
$$\mathbf{V}^{(3)} : \big((\mathbb{N} \nearrow \mathbb{N}) \nearrow (\mathbb{N} \nearrow \mathbb{N})\big), \quad \ldots \quad \mathbf{V}^{(\delta)} : \mathbb{N} \overset{\delta-1}{\nearrow} \mathbb{N} \ .$$

d) Let $\overset{-1}{\star} = \text{"}+\text{"} : \mathbb{N} \nearrow \mathbb{N} \nearrow \mathbb{N}$ and $\overset{0}{\star} = \text{"}\cdot\text{"} : \mathbb{N} \nearrow \mathbb{N} \nearrow \mathbb{N}$ and

$$\overset{\varepsilon}{\star} \ = \ \text{"}\circ\text{"} : \ (\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}) \nearrow (\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}) \nearrow (\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}), \quad \varepsilon = 1, \ldots, \delta$$

where $g \circ f$ denotes the polymorphic composition of $f, g : (\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}) \nearrow (\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N})$.

e) Next consider $\overset{-1}{\star}, \overset{0}{\star}, \ldots, \overset{\delta-1}{\star}$ extended pointwise to type $(\mathbb{N} \overset{\delta}{\nearrow} \mathbb{N}) \nearrow (\mathbb{N} \overset{\delta}{\nearrow} \mathbb{N}) \nearrow (\mathbb{N} \overset{\delta}{\nearrow} \mathbb{N})$. And, for $\varepsilon < \delta$, consider $\boldsymbol{\Pi}^{(\varepsilon)}$ of type $\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}$ as constant mapping of type

$$\mathbb{N} \overset{\delta}{\nearrow} \mathbb{N} \ = \ \big(\mathbb{N} \overset{\delta-1}{\nearrow} \mathbb{N}\big) \nearrow \big(\mathbb{N} \overset{\delta-2}{\nearrow} \mathbb{N}\big) \nearrow \cdots \nearrow \big(\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}\big) \nearrow \big(\mathbb{N} \overset{\varepsilon}{\nearrow} \mathbb{N}\big) \ .$$

f) Structural induction on the expression built-up according to (b) thus well-defines the semantics of Equation (10) of type $\mathbb{N} \overset{\delta}{\nearrow} \mathbb{N}$. And said family of mappings coincides with the family of order-δ polynomials according to Definition 27c).

g) This fragment of Simply Typed Lambda Calculus allows only terms in "head" normal form (10) without inner bound variables, cmp. Example 24b). Item e) employs two kinds of type coercion: i) Extending each binary operation $+, \cdot, \circ, \ldots$ pointwise from its initial $X \to X \to Y$ to a higher type $(Z \to X) \to (Z \to X) \to (Z \to Y)$. And ii) considering an element y of some type Y as constant function of type $Z \to Y$. Thus any (sub-)expression $\boldsymbol{\Xi}$ semantically induces an operator of same pure input and output type. ⌟

References

1 Aras Bacho and Martin Ziegler. Second-order parameterizations for the complexity theory of integrable functions. *arXiv CoRR*, 2506.11210, 2025. doi:10.48550/arXiv.2506.11210.

2 Patrick Baillot, Ugo Dal Lago, Cynthia Kop, and Deivid Vale. On basic feasible functionals and the interpretation method. In Naoki Kobayashi and James Worrell, editors, *Proc.II of 27th Int. Conf. Foundations of Software Science and Computation Structures (FoSSaCS2024), held as part of the European Joint Conferences on Theory and Practice of Software (ETAPS2024) in Luxembourg*, volume 14575 of *Lecture Notes in Computer Science*, pages 70–91. Springer, 2024. doi:10.1007/978-3-031-57231-9_4.

3 Eyvind Martol Briseid. Fixed point of generalized contractive mappings. *Journal of Nonlinear and Convex Analysis*, 9:2:181–204, 2008.

4 Eyvind Martol Briseid. Logical aspects of rates of convergence in metric spaces. *The Journal of Symbolic Logic*, 74(4):1401–1428, 2009. doi:10.2178/jsl/1254748697.

5 Michael Codish, Yoav Fekete, Carsten Fuhs, Jürgen Giesl, and Johannes Waldmann. Exotic Semiring Constraints. In *SMT 2012. 10th International Workshop on Satisfiability Modulo Theories*, volume 20 of *EasyChair Proceedings in Computing*, pages 88–97. EasyChair, 2013. doi:10.29007/QQVT.

6 Martin Dietzfelbinger. *Primality Testing in Polynomial Time*, volume 3000 of *Lecture Notes in Computer Science*. Springer, 2004. doi:10.1007/B12334.

7 Hugo Férée and Mathieu Hoyrup. Higher-order complexity in analysis. *HAL Archive*, July 2013. Proc 10th Int.Conf. Computability and Complexity in Analysis (CCA). URL: https://hal.inria.fr/hal-00915973.

8 Anka Gajentaan and Mark H Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

9 Mikhail Gromov. Groups of polynomial growth and expanding maps. *Publications Mathématiques de L'Institut des Hautes Scientifiques*, 53:53–78, 1981. doi:10.1007/BF02698687.

10 Emmanuel Hainry, Bruce M. Kapron, Jean-Yves Marion, and Romain Péchoux. Declassification policy for program complexity analysis. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proc. 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia*, pages 41:1–41:14. ACM, 2024. doi:10.1145/3661814.3662100.

11 Emmanuel Hainry and Romain Péchoux. Higher order interpretation for higher order complexity. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 269–285. EasyChair, 2017. doi:10.29007/1TKW.

12 Peter Hertling. Topological complexity with continuous operations. *Journal of Complexity*, 12(4):315–338, 1996. doi:10.1006/jcom.1996.0021.

13 Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996. doi:10.1137/S0097539794263452.

14 Bruce M. Kapron and Florian Steinberg. Type-two polynomial-time and restricted lookahead. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 579–588, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209124.

15 Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory*, 4(2):5:1–5:24, 2012. doi:10.1145/2189778.2189780.

16 Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Second-order linear-time computability with applications to computable analysis. In *Proc. 15th Conf. on Theory and Applications of Models of Computation (TAMC)*, pages 337–358, Berlin, Heidelberg, 2019. Springer-Verlag. doi:10.1007/978-3-030-14812-6_21.

17 Ker-I Ko. *Complexity Theory of Real Functions*, volume 3 of *Progress in theoretical computer science*. Birkhäuser / Springer, 1991. doi:10.1007/978-1-4684-6802-1.

18 Ivan Koswara, Gleb Pogudin, Svetlana Selivanova, and Martin Ziegler. Bit-complexity of classical solutions of linear evolutionary systems of partial differential equations. *Journal of Complexity*, 76:101727, 2023. doi:10.1016/j.jco.2022.101727.

19 David A. Levin and Yuval Peres. *Markov chains and mixing times*. American Mathematical Society, Providence, Rhode Island, 2009.

20 Donghyun Lim. Degrees of second and higher-order polynomials. 4th Workshop on Mathematical Logic and its Applications, March 2021. URL: http://jaist.ac.jp/event/mla2021.

21 Donghyun Lim and Martin Ziegler. Degrees of second and higher-order polynomials. *arXiv CoRR*, 2305.03439, 2023. doi:10.48550/arXiv.2305.03439.

22 Donghyun Lim and Martin Ziegler. Quantitative coding and complexity theory of *Continuous* data: Part I. *J. ACM*, 72(1):4:1–4:39, 2025. doi:10.1145/3705609.

23 George G. Lorentz. *Metric entropy and approximation*, volume 72. AMS, 1996.

24 Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Sciences*, 12(2):147–178, 1976. doi:10.1016/S0022-0000(76)80035-9.

25 Eike Neumann and Florian Steinberg. Parametrised second-order complexity theory with applications to the study of interval computation. *Theor. Comput. Sci.*, 806:281–304, 2020. doi:10.1016/j.tcs.2019.05.009.

26 Jürgen Richter-Gebert, Bernd Sturmfels, and Thorsten Theobald. First steps in tropical geometry. *Idempotent Mathematics and Mathematical Physics*, 377:289–317, 2005.

27 Jon G. Riecke and Ramesh Subrahmanyam. Conditions for the completeness of functional and algebraic equational reasoning. *Mathemat.Structures Computer Science*, 9(6):651–685, 1999. doi:10.1017/S0960129599002807.

28 Ernst Specker, Norbert Hungerbühler, and Micha Wasem. Polyfunctions over commutative rings. *J. Algebra and Applications*, 23(01):2450014, 2024. doi:10.1142/S0219498824500142.

29 Volker Strassen. Gaussian elimination is not optimal. *Numer.Mathematik*, 13:354–356, 1969.

30 Mike Townsend. Complexity for type-2 relations. *Notre Dame Formal Logic*, 31(2):241–262, 1990. doi:10.1305/NDJFL/1093635419.

31 Martin Ziegler. Hyper-degrees of 2nd-order polynomial-time reductions. Technical report, Dagstuhl Seminar #15392, 2016. abstract §3.20. doi:10.4230/DagRep.5.9.77.

A Proof of Example 16b

Recall [12, bottom of p.336] the Trisection Algorithm 1 below, approximating $f^{-1}(y)$ up to error 2^{-n}. Traditional *bi*section in $[a;b]$ for x with $f(x)=y$ fails in case $f(\frac{a+b}{2})=y$, since test for equality is impossible reliably. *In*equalities $f(\frac{a+b}{2})<y$ and $f(\frac{a+b}{2})>y$ on the other hand can be verified reliably, subject to the promise $f(\frac{a+b}{2})\neq y$. *Tri*section instead performs both tests $f(a')<y$ and $f(b')>y$ simultaneously, for $a':=\frac{2}{3}a+\frac{1}{3}b$ and $b':=\frac{1}{3}a+\frac{2}{3}b$: Whichever test succeeds reliably, is taken to proceed either to $[a';b]$ or to $[a;b']$, respectively.

Algorithm 1 Trisection.

Trisection$\big(\, f:[0;1]\to[0;1]$ increasing bijection , $y\in[0;1]$, $n\in\mathbb{N}\,\big)\ \in\ [0;1]$
let $\mathbb{R}\ni a:=0$; **let** $\mathbb{R}\ni b:=1$;
for $\mathbb{Z}\ni k:=1$ **to** $\lceil n/\log_2(\frac{3}{2})\rceil$ **do** ▷ $f(a)\le y\le f(b)$, $b-a=(\frac{2}{3})^{k-1}$
  let $\mathbb{R}\ni a':=\frac{2}{3}a+\frac{1}{3}b$; **let** $\mathbb{R}\ni u:=f(a')$;
  let $\mathbb{R}\ni b':=\frac{1}{3}a+\frac{2}{3}b$; **let** $\mathbb{R}\ni v:=f(b')$;
  if $v>y$ **let** $b:=b'$ **xorif** $u<y$ **let** $a:=a'$ **endif**; ▷ $f(a)\le y\le f(b)$, $b-a=(\frac{2}{3})^{k}$
end for
return $\big((a+b)/2\big)$;

In Line 6 of the Pseudo-code 1, `xorif` indicates that both comparisons "$v > y$" and "$u < y$" are conducted in parallel such that, non-deterministically, precisely one of the two assignments gets executed. Since f is guaranteed strictly increasing, at least one of the two conditions holds; possibly both. To find out which one, take approximations $\tilde{u}$ to $u = f(a)$ and $\tilde{v}$ to $v = f(b)$ up to sufficient precision. For that in turn take approximations to a and b up to sufficient precision. This incurs total bit-cost $\mathcal{O}(\nu(m) + \mu(\nu(m)+2))$, as argued in the sequel:

Indeed, ν being a modulus of continuity of f^{-1} implies $v - u > 2^{-\nu(m)}$ since $f^{-1}(v) - f^{-1}(u) = b' - a' = (\frac{2}{3})^{k+1} > 2^{-m}$ for $m := \lceil (k+1) \cdot \log_2(\frac{3}{2}) \rceil$. Hence dyadic approximations $\tilde{u}$ to $u = f(a')$ and $\tilde{v}$ to $v = f(b')$ and $\tilde{y}$ to y, all up to error $2^{-\nu(m)-3}$, will confirm at least one of the two inequalities

$$\tilde{v} > \tilde{y} + 2^{-\nu(m)-2}, \qquad \tilde{u} < \tilde{y} - 2^{-\nu(m)-2}$$

satisfied as witnesses to $v > y$ or $u < y$, respectively; to which in turn approximations to a', b' suffice up to error $2^{-\mu(\nu(m)+2)}$. Hence Lines 6 and 4 and 5 incur bit-cost

$$\mathcal{O}\Big(\nu(m) \;+\; \mu\big(\nu(m)+2\big)\Big) \;\leq\; \mathcal{O}\Big(\nu(n+2) \;+\; \mu\big(\nu(n+2)+3\big)\Big) \;.$$

And these lines are repeated $\mathcal{O}(n)$ times according to Line 3. ◀

B Selected Further Deferred Proofs

Proof of Example 19. Calculate

$$\begin{aligned}
\overline{\mathbf{\Pi}}\big(m \mapsto a \cdot m^d, n\big) &= \big(a^{2d+2} + a^{d+1}\big) \cdot n^{2d^2} \\
\overline{\mathbf{\Xi}}\big(m \mapsto a \cdot m^d, n\big) &= 2a^{2d+1} \cdot n^{2d^2} \\
\overline{\mathbf{\Pi}}\big(m \mapsto m^d + c, n\big) &= \big((n^d + c)^d + c\big)^2 \;+\; \big(n^{2d} + c\big)^d \\
&\approx \big(n^{d^2} \;+\; cdn^{d(d-1)} \;+\; \ldots\big)^2 \;+\; N^{2d^2} \;+\; cdn^{2d\cdot(d-1)} \;+\; \cdots \\
&\approx 2n^{2d^2} \;+\; 2cdn^{d^2+2d\cdot(d-1)} \;+\; \cdots \\
\overline{\mathbf{\Xi}}\big(m \mapsto m^d + c, n\big) &= 2 \cdot (n^d + c)^{2d} \;+\; 2c \\
&\approx 2n^{2d^2} \;+\; \mathbf{2}(2d)cn^{d(2d-1)} \;+\; \cdots
\end{aligned}$$

where "≈" means first (few) terms in the Taylor expansion w.r.t. $n \to \infty$. ◀

Proof of Example 14iv+v.

iv) By definition of $\mathbf{\Pi}$, running $\mathcal{M}^{\varphi}(\vec{x})$ makes at most $\overline{\mathbf{\Pi}}(|\varphi|, |\vec{x}|)$ steps. In particular its output $\vec{y}$ has length at most $\overline{\mathbf{\Pi}}(|\varphi|, |\vec{x}|)$. By definition of $\mathbf{\Xi}$, $\mathcal{N}^{\varphi}(\vec{y})$ in turn makes at most $\overline{\mathbf{\Xi}}(|\varphi|, |\vec{y}|)$ steps.

v) $\mathcal{M}^{\psi}(\vec{x})$ makes at most $\mathbf{\Pi}(|\psi|, |\vec{x}|)$ steps and in particular makes at most that many queries "$G(\varphi, \vec{y}) = \mathbf{?}$" to oracle $\psi = \mathcal{N}^{\varphi}$, each of bounded length $m = |\vec{y}| \leq \mathbf{\Pi}(|\psi|, |\vec{x}|)$. Similarly, $\psi = \mathcal{N}^{\varphi}$ answering any such query $\vec{y}$ takes time at most $\mathbf{\Xi}(|\varphi|, m)$ and in particular returns an answer $\vec{z} = \psi(\vec{y})$ of length $|\vec{z}| \leq \mathbf{\Xi}(|\varphi|, m)$: hence $|\psi|(m) \leq \mathbf{\Xi}(|\varphi|, m)$. ◀

Proof of Example 17iii. For one direction, suppose that π satisfies

$$\forall m\, \forall M < 2^m : \quad \pi(M) \;<\; 2^{q(m)} \;. \tag{11}$$

Then $\pi^{-1}\big[2^{q(m)}; \infty\big) \subseteq \big[2^m; \infty\big)$ and therefore

$$\sum\nolimits_{L \geq 2^{q(m)}} |z_{\pi^{-1}(L)}|^p \;\leq\; \sum\nolimits_{K \geq 2^m} |z_K|^p \;\leq\; 2^{-pn}$$

for $m := \sigma(n)$ by hypothesis. Hence $\tau := q \circ \sigma$ is a (not necessarily minimal, *i.e.* an upper bound on the) modulus of convergence of the image sequence.

Conversely suppose $\tau = \mathbf{\Pi}(\sigma, \cdot)$ is some (upper bound on the minimal) modulus of convergence of the image sequence whenever σ is for the original sequence: Since $\mathbf{\Pi}$ is monotone, also σ need not be minimal. Fix $m \geq 3$ and note that $\sigma :\equiv m$ constitutes a joint modulus of convergence to all sequences $\bar{z}^{(M)} = (0, \ldots, 0, \mathbf{1}, 0, \ldots) \in \ell^p$ with value 1 only at one index $M < 2^m$. The image of $\bar{z}^{(M)}$ is the sequence $\bar{z}^{(\pi(M))}$, which by hypothesis satisfies $\frac{1}{2} \geq \sum_{L \geq 2^{\tau(1)}} \left| z^{(M)}_{\pi^{-1}(L)} \right|^p \geq 1$ unless $\pi(M) < 2^{\tau(1)} = 2^{q(m)}$ for all $M < 2^m$, with $q(m) := \mathbf{\Pi}(\lambda n.m, 1)$ a first-order polynomial according to Proposition 12c). ◀

A Finer View of the Parameterized Landscape of Labeled Graph Contractions

Yashaswini Mathur
Indian Institute of Science Education and Research Bhopal, India

Prafullkumar Tale
Indian Institute of Science Education and Research Pune, India

Abstract

We study the Labeled Contractibility problem, where the input consists of two vertex-labeled graphs G and H, and the goal is to determine whether H can be obtained from G via a sequence of edge contractions.

Lafond and Marchand [WADS 2025] initiated the parameterized complexity study of this problem, showing it to be W[1]-hard when parameterized by the number k of allowed contractions. They also proved that the problem is fixed-parameter tractable when parameterized by the tree-width tw of G, via an application of Courcelle's theorem resulting in a non-constructive algorithm.

In this work, we present a constructive fixed-parameter algorithm for Labeled Contractibility with running time $2^{\mathcal{O}(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$. We also prove that unless the Exponential Time Hypothesis (ETH) fails, it does not admit an algorithm running in time $2^{o(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$. This result adds Labeled Contractibility to a small list of problems that admit such a lower bound and matching algorithm.

We further strengthen existing hardness results by showing that the problem remains NP-complete even when both input graphs have bounded maximum degree. We also investigate parameterizations by $(k + \delta(G))$ where $\delta(G)$ denotes the degeneracy of G, and rule out the existence of subexponential-time algorithms. This answers question raised in Lafond and Marchand [WADS 2025]. We additionally provide an improved FPT algorithm with better dependence on $(k + \delta(G))$ than previously known. Finally, we analyze a brute-force algorithm for Labeled Contractibility with running time $|V(H)|^{\mathcal{O}(|V(G)|)}$, and show that this running time is optimal under ETH.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Labeled Contraction, ETH Lower-bound, Treewidth, NP-hard

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.43

Related Version *Full Version*: https://arxiv.org/abs/2510.06102

Funding *Prafullkumar Tale*: INSPIRE Faculty Fellowship by DST, Govt of India.

1 Introduction

Graph modification problems are a foundational topic in algorithmic graph theory. A general formulation of the $\mathcal{F}$-M-Modification problem asks: given a graph G and a fixed set M of allowed modification operations (such as vertex deletion, edge addition/deletion, or edge contraction), can G be transformed into a graph in class $\mathcal{F}$ using at most k operations from M.

It is well known that the $\mathcal{F}$-M-Modification problem is NP-hard for many natural graph classes $\mathcal{F}$ when M consists of vertex or edge deletions [42, 54]. Hardness results for edge contraction were first shown by Watanabe et al. [53] and Asano and Hirata [4]. It is noteworthy that while the problem becomes trivially solvable when M includes only vertex or edge deletions and $\mathcal{F}$ is a singleton, this is not the case for edge contractions. In fact, Brouwer and Veldman [9] established that the problem remains NP-complete even when $\mathcal{F}$ is a singleton containing a small graph, such as a cycle or a four-vertex path. This difference highlights the increased algorithmic complexity introduced by edge contractions compared to vertex or edge deletions.


© Yashaswini Mathur and Prafullkumar Tale;
licensed under Creative Commons License CC-BY 4.0



Graph modification problems have played a central role in the development of parameterized complexity and the design of key algorithmic techniques. A representative list of relevant works includes [7, 8, 11–14, 17, 23–25, 27, 29]. For a comprehensive overview, the survey by Crespelle et al. [18] provides an in-depth discussion of modification problems involving edge deletion, editing, and completion.

A series of recent papers have studied the parameterized complexity of $\mathcal{F}$-Contraction for various graph classes $\mathcal{F}$, including paths and trees [35], generalizations and restrictions of trees [1, 2], cactus graphs [40], bipartite graphs [34, 36], planar graphs [33], grids [50], cliques [10], split graphs [3], chordal graphs [44], bicliques [47], and degree-constrained graph classes [5, 32, 51].

Recently, Lafond and Marchand [41] initiated the parameterized study of edge contraction problems in *uniquely labeled graphs*, where each vertex has a different label. They introduced the following problem:

Labelled Contractibility
Input: Two vertex-labeled graphs G and H with $V(H) \subseteq V(G)$.
Question: Is H a labeled contraction of G?

See Definition 6 in Section 2 for formal definitions.

They also introduced a generalization, motivated by applications in phylogenetic networks [46], which are rooted acyclic directed graphs with labeled leaves representing evolutionary histories:

Maximum Common Labeled Contraction
Input: Two vertex-labeled graphs G and H, and an integer k.
Question: Do there exist labeled contraction sequences S_1 and S_2 such that $G/S_1 = H/S_2$ and $|S_1| + |S_2| \leq k$?

This second problem is closely related to the first. As shown in [41], H is a labeled contraction of G if and only if there exists a common contraction of both G and H to a labeled graph H' of size at least $|V(H)|$. This equivalence allows complexity results to be transferred between the two problems. The same work provides additional applications and results.

Lafond and Marchand [41] showed that Labeled Contractibility is W[1]-hard when parameterized by the natural parameter $k = |V(G) \setminus V(H)|$, the number of allowed edge contractions. This motivates the exploration of structural parameters. They established that the problem admits a fixed-parameter tractable (FPT) algorithm when parameterized by the treewidth tw of the input graph G by invoking a variation of Courcelle's theorem [26, Theorem 11.73]. However, this approach yields a non-elementary dependence on the parameter.

Our first contribution is a constructive dynamic programming algorithm with a running time that is singly exponential in tw^2. We complement this with a matching lower bound under the Exponential Time Hypothesis (ETH):

▶ **Theorem 1.** *Labeled Contractibility*
- *admits an algorithm running in time* $2^{\mathcal{O}(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$*; but*
- *does not admit an algorithm running in time* $2^{o(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$*, unless the ETH fails.*

We note that this lower bound also holds when parameterized by the pathwidth, which is a larger parameter. Most known FPT-algorithms that use dynamic programming on tree decompositions have running times, and matching lower bounds, of the form $2^{\mathcal{O}(\mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ [19, Chapters 7,13] or $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})} \cdot n^{\mathcal{O}(1)}$ [21, 43]. For a relatively rare set of problems exhibits

a double-exponential dependence on treewidth, see [16, 30, 52] and references within. To our knowledge, the only problems with single-exponential-but-polynomial dependencies on pathwidth are mentioned in [20, 48, 49]. Similar results parameterized by vertex cover number can be found in [3, 6, 15, 31].

Lafond and Marchand [41] observed that smaller parameters such as degeneracy do not yield tractable results. Specifically, it is known that the Maximum Common Labeled Contraction problem is NP-hard even when both input graphs have constant maximum degree (and thus constant degeneracy), and that Labeled Contractibility is NP-hard for graphs of bounded degeneracy [45, Theorem 8]. We strengthen this latter result by proving hardness under an even stricter constraint:

▶ **Theorem 2.** *Labeled Contractibility is* NP*-hard even when both G and H have bounded maximum degree.*

These results suggest that tractability is unlikely when parameterizing solely by the solution size k or the degeneracy $\delta(G)$. Nevertheless, Lafond and Marchand [41] showed the problem becomes tractable for the combined parameter $(k + \delta(G))$, yielding an algorithm with a running time of $(\delta(G) + 2k)^k \cdot n^{\mathcal{O}(1)}$. They asked whether a subexponential-time algorithm exists for this parameterization, a question we resolve negatively:

▶ **Theorem 3.** *Labeled Contractibility does not admit an algorithm running in time $2^{o(|V(G)|+|E(G)|)}$ even when both G and H have bounded degeneracy, unless the* ETH *fails.*

This theorem rules out the possibility of a subexponential-time algorithm parameterized by $(k + \delta(G))$ under the assumption of the ETH. We also provide the following algorithm, which improves upon previous work:

▶ **Theorem 4.** *Labeled Contractibility can be solved in time $(\delta(H) + 1)^k \cdot |V(G)|^{\mathcal{O}(1)}$, where k is the solution size and $\delta(H)$ is the degeneracy of H.*

This algorithm improves upon the $(\delta(H) + k)^k \cdot n^{\mathcal{O}(1)}$ algorithm from [41, Section 3.3]. The previous result relied on the bound $\delta(H) \leq \delta(G) + k$. We refine this with a tighter upper bound: $\delta(H) \leq \delta(G) \cdot \frac{|V(G)|}{|V(G)|-k}$. For cases where $|V(G)| \geq (1 + \epsilon) \cdot k$, this bound yields $\delta(H) \leq \delta(G) \cdot c_\epsilon$, where ϵ and c_ϵ are absolute constants. This substantially improves the previous analysis, leading to a more efficient algorithm under these conditions.

Finally, as our last contribution, we analyze a brute-force algorithm for Labeled Contractibility and establish its optimality under the ETH. Notably, our analysis also applies to the more general Maximum Common Labeled Contraction problem.

▶ **Theorem 5.** *Labeled Contractibility*
- *admits an algorithm running in time $|V(H)|^{\mathcal{O}(|V(G)|)}$; but*
- *does not admit an algorithm running in time $|V(H)|^{o(|V(G)|)}$, unless the* ETH *fails.*

Organization. We adopt standard notation from graph theory and parameterized complexity, as outlined in Section 2. In Section 3, we present a dynamic programming algorithm based on tree decompositions and establish a matching lower bound. Specifically, we provide a parameter-preserving reduction from the Sub-Cubic Partitioned Vertex Cover problem [3], which proves Theorem 1. Section 4 proves Theorem 2 by a reduction from Positive-Not-All-Equal-SAT, showing NP-hardness even for graphs of bounded maximum degree. Section 5 is dedicated to a conditional lower bound under the ETH. Using a reduction from 1-in-3-SAT, we prove this result even when both input graphs

have bounded degeneracy. We also present an improved FPT algorithm for the combined parameter $(k + \delta(G))$. These results together establish Theorems 3 and 4. Section 6 analyzes a brute-force algorithm and establishes its optimality under the ETH. This is achieved via a reduction from the Cross Matching problem [28] and establishes Theorem 5.

For more comprehensive details, including the full proofs and additional technical arguments, we refer the reader to the longer version of this paper.

2 Preliminaries

Throughout this work, we consider uniquely labeled graphs. Following the convention in [41], we refer to vertices directly by their labels rather than using explicit labeling functions. Two labeled graphs G and H are considered *equal*, denoted $G = H$, if and only if $V(G) = V(H)$ and $E(G) = E(H)$. This is distinct from the standard notion of graph isomorphism, where a bijection $\sigma : V(G') \to V(H')$ exists such that $(u, v) \in E(G')$ if and only if $(\sigma(u), \sigma(v)) \in E(H')$. In our setting, we assume that the labels of H are a subset of those of G.

▶ **Definition 6.** *Let G be a labeled graph and $(u, v) \in E(G)$. The* labeled contraction *of the edge (u, v) is an operation that transforms G by:*

- *For every vertex $w \in N(v) \setminus N[u]$, adding the edge (u, w) to G.*
- *Removing the vertex v and all incident edges.*

The resulting graph is denoted by $G/(u, v)$.

Note that in general, $G/(u, v) \neq G/(v, u)$, as the vertex that is retained after the contraction (and thus the label) is different. The two resulting graphs are, however, isomorphic under the standard unlabeled notion. For simplicity, in the remainder of this paper, we assume all contractions are labeled and refer to them simply as contractions.

A labeled contraction sequence S on a graph G is a sequence of vertex pairs $S = ((u_1, v_1), \ldots, (u_k, v_k))$ such that for each $i \in \{0, \ldots, k-1\}$, if G^i is the graph obtained after the first i contractions (with $G^0 := G$), then $(u_{i+1}, v_{i+1}) \in E(G^i)$ and $G^{i+1} = G^i/(u_{i+1}, v_{i+1})$. The graph obtained after the full sequence is denoted by G/S. If S is not a valid contraction sequence on G, then G/S is undefined. We say that a graph H is a labeled contraction of G if there exists a contraction sequence S such that $G/S = H$. A graph M is a common labeled contraction of graphs G and H if it is a labeled contraction of both. A maximum common labeled contraction of G and H is a common labeled contraction with the largest possible number of vertices.

Consider two graphs H and G such that $V(H) \subseteq V(G)$.

▶ **Definition 7.** *A witness structure of G into H is a partition $\mathcal{W} = \{W_1, \ldots, W_{|V(H)|}\}$ of $V(G)$ into non-empty sets satisfying:*

- *For each $W_i \in \mathcal{W}$, the induced subgraph $G[W_i]$ is connected.*
- *Each W_i contains exactly one vertex from $V(H)$, called its representative vertex.*
- *For any distinct $u, v \in V(H)$, the edge $(u, v) \in E(H)$ if and only if the sets in $\mathcal{W}$ containing u and v are adjacent in G.*

There is a natural bijection between the sets in $\mathcal{W}$ and the vertices of H via their representatives. It is known that H is a labeled contraction of G if and only if a witness structure of G into H exists (see Observation 2 in [41]). Furthermore, contractions within a single witness set can be performed in any order as long as the representative vertex is preserved (Observation 3 in the same work). For any vertex $u \in V(G)$, we denote by $\mathcal{W}(u) \in \mathcal{W}$ the unique set that contains u.

3 Parameterized by Treewidth

In this section, we present the proof of Theorem 1 which we restate.

▶ **Theorem 1.** *Labeled Contractibility*
- *admits an algorithm running in time* $2^{\mathcal{O}(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$*; but*
- *does not admit an algorithm running in time* $2^{o(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$*, unless the ETH fails.*

In the first subsection, we describe the algorithmic result by presenting a dynamic programming algorithm on tree decomposition. In the second subsection, we establish a matching conditional lower bound.

3.1 Algorithmic Result

In this subsection we will present a dynamic programming (DP) based algorithm for the Labeled Contraction problem when parameterized by treewidth of the input graph. For the sake of completeness, we start with the standard definitions.

▶ **Definition 8** ([19]). *A* tree decomposition *of a graph G is a pair $(\mathcal{T}, \{X_t\}_{t \in V(\mathcal{T})})$, where $\mathcal{T}$ is a rooted tree and each node $t \in V(\mathcal{T})$ is associated with a set $X_t \subseteq V(G)$, referred to as a* bag*, satisfying the following conditions:*
- *For every vertex $v \in V(G)$, the set $\{t \in V(\mathcal{T}) \mid v \in X_t\}$ is nonempty and induces a connected subtree of $\mathcal{T}$;*
- *For every edge $\{u, v\} \in E(G)$, there exists a node $t \in V(\mathcal{T})$ such that $\{u, v\} \subseteq X_t$.*

The width *of a tree decomposition is* $\max_{t \in V(\mathcal{T})} |X_t| - 1$*. The* treewidth *of G, denoted $\mathsf{tw}(G)$, is the minimum width over all valid tree decompositions of G.*

▶ **Definition 9** ([19]). *A* nice tree-decomposition *of a graph G is a rooted tree-decomposition $(\mathcal{T}, \{X_t\}_{t \in V(\mathcal{T})})$ where $\mathcal{T}$ is rooted at r with $X_r = \emptyset$, and each node of $\mathcal{T}$ belongs to one of the following types:*
- *A leaf node t is a leaf of $\mathcal{T}$ with $X_t = \emptyset$.*
- *An introduce vertex node t has one child t_1 such that $X_t = X_{t_1} \cup \{x\}$, for some $x \notin X_{t_1}$.*
- *A forget node t has one child t_1 such that $X_t = X_{t_1} \setminus \{x\}$, where $x \in X_{t_1}$.*
- *An introduce edge node t has one child t' with $X_t = X_{t'}$, and is labeled with an edge $\{u, v\} \in E(G)$ such that $\{u, v\} \subseteq X_t$. We say that the edge $\{u, v\}$ is introduced at t;*
- *A join node t has two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.*

Without loss of generality, we assume that a nice tree-decomposition of width tw is provided. If not, one can construct an optimum tree decomposition in time $2^{\mathcal{O}(\mathrm{tw}^2)} n^{\mathcal{O}(1)}$ [39].

Recall that we denote $G \cup H$ as the graph with vertex set $V(G)$ and edge set $E(G) \cup E(H)$. The following lemma enables us to work with tree decompositions of $G \cup H$ instead of just G, which is crucial for our formulation.

▶ **Lemma 10** ([41], Lemma 13). *If H is a contraction of G, then $\mathsf{tw}(G \cup H) \leq 2\mathsf{tw}(G)$.*

Henceforth, we fix a nice tree decomposition $\mathcal{T}$ of $G \cup H$. This choice is crucial because it ensures that for any edge $(u, v) \in E(H)$, there exists some bag X_t such that $u, v \in X_t$, a condition that is not guaranteed in a tree decomposition of G alone. For each node $t \in \mathcal{T}$, we define G_t and H_t to be the subgraphs of G and H, respectively, induced by the union of all bags in the sub-tree of $\mathcal{T}$ rooted at t.

We now introduce the key definitions for the DP-state at a node t of the tree-decomposition to convey the main ideas of the algorithm. The formal proof follows.

3.1.1 Witness Structures and Functional Representation

Assume that H is a contraction of G, and let $\mathcal{W} = \{W_1, \ldots, W_{|V(H)|}\}$ denote a witness structure for this contraction. We represent $\mathcal{W}$ functionally using a pair of auxiliary functions: a *parent function* and an *originator function*.

▶ **Definition 11** (Parent function). *Given a witness structure $\mathcal{W}$, a function $\sigma' : V(G) \to V(G)$ is a* parent function *with respect to $\mathcal{W}$ if there exists a collection of rooted trees $\{T_i\}_{i=1}^{k}$, where each T_i is a spanning tree of $G[W_i]$ rooted at the unique vertex of $W_i \cap V(H)$, such that:*

1. *For all $u \in W_i \setminus V(H)$, $\sigma'(u)$ is the parent of u in T_i; and*
2. *For the unique vertex $u \in W_i \cap V(H)$, $\sigma'(u) = u$.*

▶ **Definition 12** (Originator function). *Given a witness structure $\mathcal{W}$, a function $\tau' : V(G) \to V(H)$ is an* originator function *with respect to $\mathcal{W}$ if for every $u \in W_i$, $\tau'(u)$ is the unique vertex in $W_i \cap V(H)$.*

Note that any valid originator function τ' defines a partition of $V(G)$ into the witness sets $\mathcal{W}$. Also, for any vertex $u \in V(G)$, there exists a bag that contains both u and $\sigma'(u)$, but not necessarily $\tau'(u)$.

A pair (σ', τ') is said to be *consistent* if for every $u \in V(G)$ $\tau'(u) = \tau'(\sigma'(u))$, i.e. u and its parent $\sigma'(u)$ belong to the same witness set. In the remainder of the discussion, we assume all parent-originator pairs to be consistent unless stated otherwise. Although τ' can be derived from σ' by following parent pointers up to a vertex in $V(H)$, we maintain both functions explicitly for algorithmic clarity.

▶ **Definition 13** (Witness impression). *Every witness structure $\mathcal{W}$ corresponds to pair (σ', τ'), called its* witness-impression*, where σ' is a parent function and τ' is the corresponding originator function.*

▶ **Definition 14** (Seed impression). *Let (σ', τ') be an impression of a witness structure. A pair of partial functions (σ, τ), with $\sigma : V(G_t) \to V(G_t) \cup \{\star\}$ and $\tau : V(G_t) \to V(H_t) \cup \{\star\}$, is a* seed impression *of (σ', τ') at t if:*

- *For every vertex $u \in V(G_t) \setminus X_t$, $\sigma(u) = \sigma'(u)$ and $\tau(u) = \tau'(u)$.*
- *For a vertex $u \in X_t$, if $\sigma'(u) \in V(G_t)$ then $\sigma(u) = \sigma'(u)$ otherwise $\sigma(u) = \star$. The function τ is derived from σ by following parent pointers up to either a vertex in $V(H)$ or $\star$.*

Note that (σ, τ) is not merely the restriction of (σ', τ') to $V(G_t)$, since a vertex u such that $\sigma'(u) \notin V(G_t)$ might satisfy $\tau'(u) \in V(G_t)$, but we still assign $\tau(u) = \star$. Note that in Figure 1, $\tau'(u_1) \in G_t$ but since $\sigma'(u_1) \notin G_t$ we assign $\tau(u_1) = \star$.

We now classify each vertex in the bag X_t into one of three types, according to the seed impression (σ, τ):

▶ **Definition 15** (Types of vertices). *Let $t \in V(\mathcal{T})$ be a node with bag X_t, and let (σ, τ) be a seed impression at t. Then each vertex $u \in X_t$ is assigned one of the following types:*

- Requester: *$\sigma(u) \in V(G_t)$ and $\tau(u) \in V(H_t)$ (i.e., both are defined); u is ready to request adjacency requirements in H.*
- Dependent: *$\sigma(u) \in V(G_t)$ but $\tau(u) = \star$; u has been contracted into its parent and depends on the parent node's decision regarding the partition.*
- Explorer: *$\sigma(u) = \star$ (implying $\tau(u) = \star$); u is still exploring who its parent will be.*

We classify the pair of requester vertices into the following three parts.

▶ **Definition 16** (Requester pair classification). *Let $u, v \in X_t$ be requester vertices with $\tau(u) = h_u$, $\tau(v) = h_v$. We say the pair (u, v) is*

- invalid *(inv) if $(h_u, h_v) \notin E(H)$ but $\tau^{-1}(h_u), \tau^{-1}(h_v)$ are adjacent in G_t,*
- unsatisfied *(unsat) if $(h_u, h_v) \in E(H)$ and $\tau^{-1}(h_u), \tau^{-1}(h_v)$ are not adjacent in G_t, and*
- satisfied *(sat), otherwise.*

For a vertex $u \in V(G)$, we define the set $\mathsf{des}(u) \subseteq V(G)$ of *descendants* of u with respect to a function $\sigma' : V(G) \to V(G)$ as follows:

$$\mathsf{des}(u) = \left\{ v \in V(G) \;\middle|\; \text{there exists an integer } \ell \geq 0 \text{ such that } \sigma'^{(\ell)}(v) = u \right\},$$

where $\sigma'^{(0)}(v) = v$, and for $\ell \geq 1$, $\sigma'^{(\ell)}(v) = \sigma'(\sigma'^{(\ell-1)}(v))$. That is, $\mathsf{des}(u)$ consists of all vertices v for which repeated applications of σ' eventually reach u, including u itself.

▶ **Definition 17** (Pseudo-adjacency). *Let u, v be two vertices in X_t. Then, we say u and v are* pseudo-adjacent *if sets $\mathsf{des}(u)$ and $\mathsf{des}(v)$ are adjacent in G_t.*

3.1.2 Signatures and Dynamic Programming States

Suppose the graph G admits an H-witness structure $\mathcal{W}$, and let (σ', τ') be the corresponding witness-impression as specified in Definition 13. Let T be a tree decomposition of the graph $G \cup H$, and let t be a node of T.

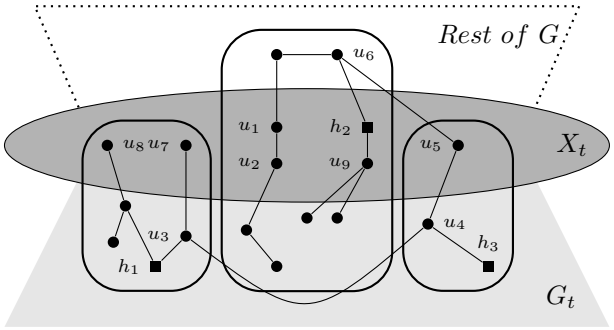
▶ **Definition 18** (Partial Solution). *We define a* partial solution *at node t as a tuple $P = (\sigma, \tau, \mathcal{R}, \mathcal{A})$, with the following components:*

- *(σ, τ) is a seed impression of the witness-impression (σ', τ'), restricted to the bag X_t at t, as specified in Definition 14,*
- *$\mathcal{R} \subseteq R \times R \times \{\mathsf{inv}, \mathsf{sat}, \mathsf{unsat}\}$ is a set of labeled pairs, where $R \subseteq X_t$ denotes the set of requester vertices contained in X_t as per Definition 15. Each element $(u, v, \ell) \in \mathcal{R}$ represents a label $\ell \in \{\mathsf{inv}, \mathsf{sat}, \mathsf{unsat}\}$ assigned to the pair (u, v), as per Definition 16,*
- *$\mathcal{A} \subseteq X_t \times X_t$ records unordered pairs of vertices in X_t that are* pseudo-adjacent *to each other, as per Definition 17.*

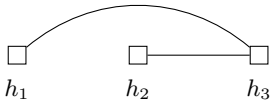
We now refer to Figure 1 to provide intuition for the above definitions. The figure shows the graph H together with a part of the tree decomposition of $G \cup H$ (we do not explicitly draw G). The rounded outlines indicate the witness structure for the contraction, whenever it exists: each group contains exactly one node of H, representing the collection of vertices that contract to it. Note that $\sigma(u_2) = u_1$, $\tau(u_2) = \star$, and $\tau(u_1) = \sigma(u_1) = \star$. Since the edge (u_3, u_4) is an edge we processed earlier in the DP, we may store $(u_7, u_5, \mathsf{sat}) \in \mathcal{R}$ for X_t. On the other hand, because the edge (u_6, u_5) has not yet been introduced, the edge (h_2, h_3) does not appear in the current structure, so we have $(u_2, u_5, \mathsf{unsat})$. Furthermore, observe that (u_8, u_5) and (u_7, u_5) belong to $\mathcal{A}$, whereas $(u_9, u_5) \notin \mathcal{A}$. In this way, all relevant sets are defined for the dynamic programming.

We now provide a formal definition for the same.

Note that, as it is standard in the case of dynamic programming on tree decomposition, we will not compute all the partial solution at node t but rather just store unique *signatures* of all the partial solutions at t. The signature S of a partial solution $P = (\sigma, \tau, \mathcal{R}, \mathcal{A})$ is $(\sigma|_{X_t}, \tau|_{X_t}, \mathcal{R}, \mathcal{A})$.



(a) Illustration of G_t and X_t.



(b) Graph H.

Figure 1 Depicting definitions relevant to the algorithm.

3.1.2.1 States of the Dynamic Programming Algorithms

▶ **Definition 19** (Signature). *A state of dynamic programming table will be a tuple* $S = (\sigma, \tau, \mathcal{R}, \mathcal{A})$ *such that*

- $\sigma : X_t \to X_t \cup \{\star\}$ *(which will emulate a parent function),*
- $\tau : X_t \to V(H_t) \cup \{\star\}$ *(which will emulate an originator function),*
- $\mathcal{R} \subseteq R \times R \times \{\mathsf{inv}, \mathsf{sat}, \mathsf{unsat}\}$ *is a set of labeled pairs representing a label* $\ell \in \{\mathsf{inv}, \mathsf{sat}, \mathsf{unsat}\}$ *assigned to the pair* (u, v)*, as per Definition 16, and*
- $\mathcal{A}$ *is the collection of pair of vertices in* X_t *that are pseudo-adjacent with each other.*

The condition on functions ensures that at present σ and τ are valid candidate to be a parent function and an originator function.

For a node t in tree decomposition T, we construct a dynamic-programming algorithm that sets the value $\mathbf{d}[S, t]$ to `True` if and only if S is a signature of some partial solution P at node t. Otherwise, we set it to `False`. Initially, all entries are set to `False`. For any signature $S = (\sigma, \tau, \mathcal{R}, \mathcal{A})$, if $\mathcal{R}$ contains an entry of the type (u, v, inv) then $\mathbf{d}[S, t]$ is set to `False` and it will never be changed.

We remark that the technical challenge in designing the dynamic programming algorithm was to find an appropriate, concise expression for the partial solution, which was discussed above. At any node t of the tree decomposition, the number of distinct signatures that can be stored is bounded by $2^{\mathcal{O}(\mathsf{tw}^2)}$, where tw denotes the width of the decomposition. This bound arises from the fact that the signature must encode information pertaining to all pairs of vertices in the bag X_t. The specific process to update the dynamic programming table is a standard exhaustive enumeration over compatible signatures at the child node(s) and applying the corresponding update rule. For full details on the update rules and their proof of correctness, we refer the reader to the longer version of this paper. The time required to process each node is polynomial in the number of signatures. Consequently, the overall running time of the algorithm is bounded by $2^{\mathcal{O}(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$. This concludes the proof of the algorithmic result mentioned in Theorem 1.

3.2 Conditional Lower Bound

In this section, we demonstrate that unless (ETH) fails, Labeled Contraction does not admit an algorithm running in time $2^{o(\mathsf{tw}^2)} \cdot |V(G)|^{\mathcal{O}(1)}$, where tw is the treewidth of the input graph G. To obtain the lower bound, we reduce from a special case of Vertex Cover on sub-cubic graphs, defined in [3] as follows:

Sub-Cubic Partitioned Vertex Cover (Sub-Cubic PVC)
Input: A sub-cubic graph G; an integer t; for $i \in [t]$, an integer $k_i \geq 0$; a partition $\mathcal{P} = \{C_1, \ldots, C_t\}$ of $V(G)$ such that $t \in \mathcal{O}(\sqrt{|V(G)|})$ and for all $i \in [t]$, C_i is an independent set and $|C_i| \in \mathcal{O}(\sqrt{|V(G)|})$. Furthermore, for $i, j \in [t]$, $i \neq j$, $|E(G[C_i \cup C_j]) \cap E(G)| = 1$.
Question: Does G have a vertex cover X such that for all $i \in [t]$, $|X \cap C_i| \leq k_i$?

▶ **Proposition 20** (Theorem 3.9 in [3]). *Sub-Cubic PVC does not admit an algorithm running in time $2^{o(n)}$, unless the ETH fails.*

We now prove the lower bound in Theorem 1.

Let $(G, \mathcal{P} = \{C_1, C_2, \ldots, C_t\}, k_1, \ldots, k_t)$ be an instance of the Sub-Cubic PVC problem, where $n = |V(G)|$. We construct an instance (G', H') of the Labeled Contraction problem as follows.

Construction of G'. We begin by setting $V(G') \supseteq V(G)$, i.e., all vertices of G are copied into G' (without copying any edges). For each partition $C_i \in \mathcal{P}$, we introduce a set S_i consisting of $\binom{|C_i|}{k_i+1}$ new vertices in G', where each vertex $s \in S_i$ corresponds uniquely to a subset $C_i' \subseteq C_i$ of size exactly $k_i + 1$. For each such vertex $s \in S_i$, we add edges between s and every vertex in the subset C_i. Additionally, for every $C_i \in \mathcal{P}$, we introduce three auxiliary vertices $\{x_i, y_i, z_i\}$ in G'. Each of these vertices is connected to every vertex in C_i. Moreover, we add edges (z_i, x_i) and (z_i, y_i). Each vertex $s \in S_i$ is connected to the corresponding vertex x_i.

We now encode the edges in G. Consider a pair $i, j \in [t]$ and an edge $(u, v) \in E(G)$ with $u \in C_i$ and $v \in C_j$. By the problem definition, such an edge exists. We add edges (u, x_j) and (v, x_i) to G'.

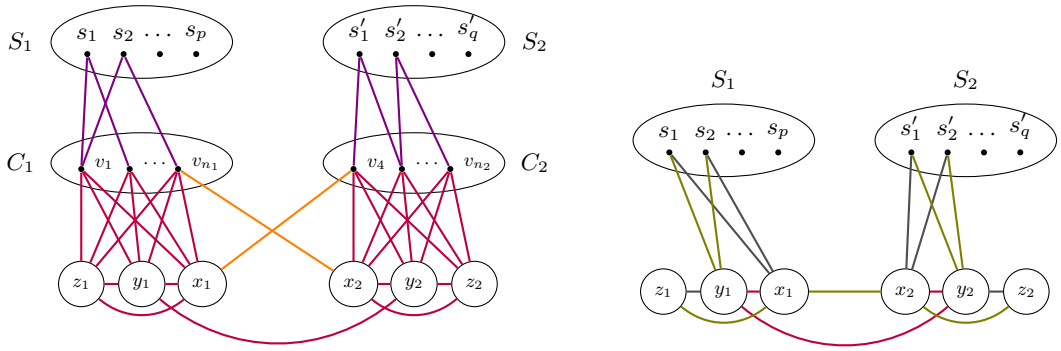
Finally, the set $\{y_1, y_2, \ldots, y_t\}$ is made into a clique by adding all possible edges between them, and for every $i, j \in [t]$, we add the edge (x_i, y_j) in G'.

Construction of H'. We start constructing H' from G' by deleting all vertices in G in it. Formally, $V(H') = \left(\bigcup_{i=1}^{t} S_i\right) \cup \left(\bigcup_{i=1}^{t} \{x_i, y_i, z_i\}\right)$, that is, it includes all newly introduced vertices from the sets S_i and the auxiliary vertices x_i, y_i, z_i for each $i \in [t]$, but does not include any vertex from the original graph G. The edge set of H' is defined as follows. First, a clique is formed on the vertex set $\{y_1, y_2, \ldots, y_t\} \cup \{x_1, x_2, \ldots, x_t\}$, adding all edges between these vertices. Additionally, for every $i \in [t]$, we include the edges (x_i, z_i) and (y_i, z_i). Finally, for every $i \in [t]$ and each vertex $s \in S_i$, we add the edge (s, y_i).

This concludes the construction. We refer the reader to Figure 2 for an illustration of the reduction. We now prove the correctness of the reduction.

($\Rightarrow$) Suppose we are given a solution to the Sub-Cubic PVC instance, that is, a vertex cover $X \subseteq V(G)$ such that for every $i \in [t]$, it holds that $|X \cap C_i| \leq k_i$. We now describe a contraction sequence on G' that yields H'. For each partition $C_i \in \mathcal{P}$, we proceed as follows: For every vertex $a \in X \cap C_i$, we contract the edge (a, x_i). For every vertex $b \in C_i \setminus X$, we contract the edge (b, y_i).

Observe that since X is a vertex cover of G, for each pair $i \neq j \in [t]$, there exists exactly one edge in G between C_i and C_j, say $(u, v) \in E(G)$, with $u \in C_i$ and $v \in C_j$. The contractions described above will result in u being contracted to x_i or y_i, and v to x_j or y_j. Since $u \in X$ or $v \in X$, at least one of the endpoints is contracted to an x-vertex, ensuring that each pair (x_i, x_j) becomes adjacent in the resulting graph, thereby forming a clique on the set $\{x_1, x_2, \ldots, x_t\}$.



(a) Graph G'. **(b)** Graph H'.

Figure 2 Illustration of the graphs G' and H' in the reduction for the lower bound of Theorem 1.

Furthermore, for each $i \in [t]$, since $|X \cap C_i| \leq k_i$, it follows that no subset of C_i of size $k_i + 1$ is entirely mapped to x_i. Therefore, for every vertex $s \in S_i$, which corresponds to a subset of C_i of size $k_i + 1$, at least one vertex in that subset is contracted to y_i. Consequently, the edge (s, y_i) appears in the resulting graph, as required by H'. The edges (x_i, y_i) and (y_i, z_i) are preserved directly from the original construction. Thus, the contraction process yields graph H', completing the forward direction of the reduction.

($\Leftarrow$) Suppose we are given a solution to the Labeled Contraction instance, i.e., a sequence of edge contractions that transforms G' into H'. We construct a corresponding solution $X \subseteq V(G)$ to the original Sub-Cubic PVC instance as follows.

Recall that the vertex set of H' is precisely $V(G') \setminus V(G)$, which implies that every contracted edge in the solution involves a vertex from $V(G)$, more precisely vertices in C_i for each $i \in [t]$. Furthermore, observe that for each $i \in [t]$, the vertices z_i and those in S_i do not share edges with each other or with vertices in $V(G)$ outside of C_i neither in G' nor in H'. In particular, the absence of edges of the form (z_i, s) for any $s \in S_i$, and the absence of edges (z_i, x_j) or (z_i, y_j) for $i \neq j$, implies that vertices in C_i can only be either contracted to x_i or y_i.

Now, for each vertex $s \in S_i$, we note that s is adjacent to y_i in H'. In the construction of G', the vertex s was made adjacent to a subset $C'_i \subseteq C_i$ of size $k_i + 1$. In order for s to remain adjacent to y_i in H', it is necessary that at least one vertex in C'_i is contracted to y_i. Thus, no subset of C_i of size $k_i + 1$ can be entirely contracted to x_i. This ensures that at most k_i vertices from C_i are contracted to x_i. Based on this observation, we define the vertex set $X \subseteq V(G)$ by including, for each $i \in [t]$, all vertices in C_i that are contracted to x_i. From the argument above, we have $|X \cap C_i| \leq k_i$ for all $i \in [t]$.

It remains to show that X is a vertex cover of G. Observe that in the construction of G', for each pair $i, j \in [t]$, $i \neq j$, such that $E(G[C_i \cup C_j]) \cap E(G) = \{(u, v)\}$ with $u \in C_i$, $v \in C_j$, we added edges (u, x_j) and (v, x_i) to G'. Since x_i and x_j are adjacent in H', and since no edge directly connects x_i and x_j in G', at least one of the vertices u or v must be contracted to x_i or x_j, respectively. This implies that either $u \in X$ or $v \in X$, and hence X intersects every edge in $E(G)$. Therefore, X is a valid vertex cover of G. This completes the proof of correctness for the reduction.

To upper bound the treewidth of the graph G', define the set $W = \{x_i, y_i, z_i \mid i \in [t]\}$. Observe that the graph $G' - W$ consists of t vertex-disjoint subgraphs, each corresponding to a complete bipartite graph between C_i and S_i. For each $i \in [t]$, the set C_i is of size $\mathcal{O}(\sqrt{n})$. This implies that the treewidth of each $G'[C_i \cup S_i]$ is $\mathcal{O}(\sqrt{n})$. Therefore, treewidth of G' is at most $\mathsf{tw}(G') \leq |W| + \max_{i \in [t]} \mathsf{tw}(G'[C_i \cup S_i]) = \mathcal{O}(\sqrt{n})$.

Now suppose, for the sake of contradiction, that there exists an algorithm $\mathcal{A}$ that solves Labeled Contraction in time $2^{o(\mathsf{tw}(G')^2)} \cdot |V(G')|^{\mathcal{O}(1)}$. We use $\mathcal{A}$ as a subroutine to construct an algorithm $\mathcal{B}$ for solving Sub-Cubic PVC. On input an instance $(G, \mathcal{P} = \{C_1, C_2, \ldots, C_t\}, k_1, \ldots, k_t)$ of Sub-Cubic PVC with $n = |V(G)|$, algorithm $\mathcal{B}$ performs the following steps: It constructs the corresponding instance (G', H') of Labeled Contraction as described in the reduction. This step takes time $2^{\mathcal{O}(\sqrt{n})}$. Then, it invoke algorithm $\mathcal{A}$ on the instance (G', H') and return the result produced by $\mathcal{A}$.

The correctness of algorithm $\mathcal{B}$ follows directly from the correctness of the reduction. Since $\mathsf{tw}(G') = \mathcal{O}(\sqrt{n})$, the total running time of $\mathcal{B}$ is $2^{o(n)}$, which contradicts the ETH, by Proposition 20. This establishes the claimed conditional lower bound stated in Theorem 1.

4 NP-hardness When Maximum Degree is Bounded

In this section, we establish the NP-hardness of the Labeled Contractibility problem, even when both the input graphs G and H have constant maximum degree. Specifically, we present a polynomial-time reduction from a variation of Positive-Not-All-3-SAT problem to the Labeled Contractibility problem, thereby proving Theorem 2. We begin by formally defining the source problem, which is proved to be NP-complete in [22].

Positive-Not-All-(3, 4)-SAT
Input: A 3-CNF Boolean formula ψ in which every literal in each clause is positive and every variable appears in at most 4 clauses.
Question: Does there exist a truth assignment to the variables such that every clause is satisfied and, in each clause, not all three literals are assigned `True`?

Let $\{x_1, x_2, \ldots, x_n\}$ denote the set of variables, and let $\{C_1, C_2, \ldots, C_m\}$ denote the set of clauses in the formula ψ. Without loss of generality, assume that the number of clauses m satisfies $2m + 2 = 2^q$ for some integer q; let $p = 2^q$ (note that p is even).

We now describe the construction of two vertex-labeled graphs G and H corresponding to the given formula.

Construction of H. Graph H is a simple path on p vertices $\{v_1, v_2, \ldots, v_p\}$.

Construction of G. We begin by constructing a complete binary tree of depth q rooted at vertex v_1, and later add some edges among its vertices. Note that the number of leaves of this binary tree is $2^q = 2m + 2$. All but two leaves of this tree are labeled as $\{C_1, a_1, C_2, a_2, \ldots, C_m, a_m\}$, alternating between clause vertices C_j and auxiliary vertices a_j. We label it such that C_i and a_i shares a parent. Consider the remaining two leaves which share a parent. Add an edge between these two leaves, label one of them as $v_{p/2}$ and other leave with any other label (as it will be irrelevant to the construction). We additionally add edges (a_i, a_{i+1}) for all $i \in [m-1]$. Now, consider the internal vertices of the constructed binary tree rooted at v_1. Finally, along the shorted path between v_1 and $v_{p/2}$, label the vertices as $v_2, v_3, \cdots, v_{p/2-1}$ respectively. For every $j \in [p/2-1]$, let U_j be the collection of vertices that are at exactly j distance from v_1. Alternately, these are all 2^j many vertices in j^{th}-level of the complete binary tree. We add a path connecting these vertices. The internal structure of the tree, including its dummy vertices and edges, is constructed as illustrated in Figure 3.

A mirrored copy of this tree is constructed symmetrically, rooted at vertex v_p, with the corresponding leaf labels $\{C'_1, a'_1, C'_2, a'_2, \ldots, C'_m, a'_m\}$, such that each primed vertex is associated with its unprimed counterpart. Additionally, we add the labels $v_{p/2+1}, \cdots, v_p$

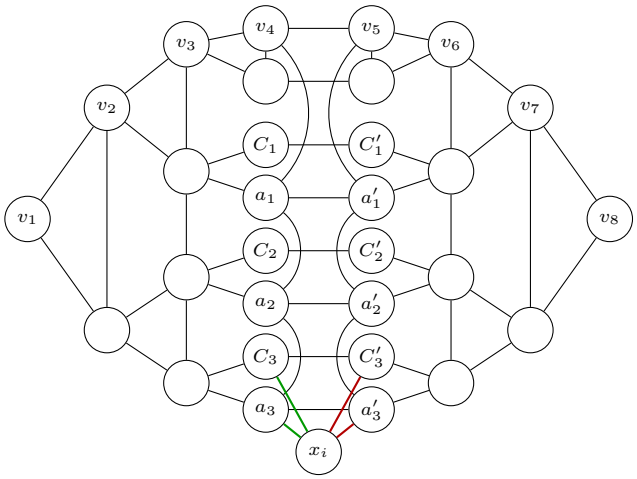


Figure 3 Example of the construction of G with $m = 3$. We only show one x_i for convenience that appears in C_3.

along the shortest path from $v_{p/2}$ to v_p. Furthermore, we add an edge (C_j, C'_j) for each $j \in [m]$, and an edge $(v_{p/2}, v_{p/2+1})$, thereby connecting the two binary trees via the central edge of the path. Finally, we add the edges $(a_1, v_{p/2})$ and $(a'_1, v_{p/2+1})$.

Variable Gadgets. For each variable x_i with $i \in [n]$, introduce a new vertex u_i. For each clause C_j in which x_i appears, connect u_i to the vertices $\{C_j, a_j, C'_j, a'_j\}$.

This concludes the description of the reduction. Note that in the final graph G, each vertex has degree at most 16.

For proof of the correctness, we refer the reader to the full version of this paper.

5 Parameterized by Solution Size Plus Degeneracy

We prove Theorem 3 and Theorem 4 in the following two subsections.

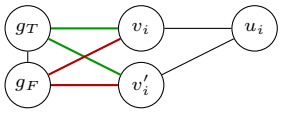
5.1 Conditional Lower Bound

We remark that most of the previous reduction about Labeled Contraction are from the variation of 3-SAT problem called Not-All-Equal-3-SAT. In this variation, the input is a 3-CNF formula and the objective is to find an assignment that sets at least one but not all literals in every clause to set to `True`. We consider the following, different version of 3-SAT as starting point of our reduction and present a simple proof from 3-SAT to get desired conditional lower bound.

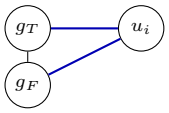
1-in-3-SAT
Input: A Boolean formula ψ in 3-CNF
Question: Does there exist a satisfying assignment such that exactly one literal is set to true in each clause?

Consider the following polynomial-time reduction from 3-SAT to 1-in-3-SAT where each clause $C^i = (x \vee \neg y \vee z)$ in the original formula is replaced by three clauses:

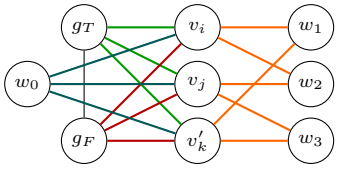
$$C^i_1 = (\neg x \vee a_i \vee b_i), \quad C^i_2 = (y \vee b_i \vee c_i), \quad C^i_3 = (\neg z \vee c_i \vee d_i),$$



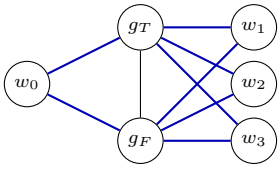
(a) Variable gadget in G.



(b) Variable gadget in H.



(c) Clause gadget in G.



(d) Clause gadget in H.

Figure 4 Illustration of the reduction. (Top) Variable gadgets in G and H for a variable (v_i). (Bottom) Clause gadgets in G and H for some clause of the form $(v_i \vee v_j \vee v'_k)$.

where a_i, b_i, c_i, d_i are fresh variables unique to the clause C^i. This reduction introduces a linear blow-up in the number of variables and clauses. Combining this reduction with the (ETH) [37] and the Sparsification Lemma [38], we obtain the following:

▶ **Proposition 21.** *Unless the ETH fails,* 1-IN-3-SAT *cannot be solved in time* $2^{o(n+m)}$*, where* n *and* m *are the number of variables and clauses in the input formula, respectively.*

We now reduce 1-IN-3-SAT to LABELED CONTRACTION, ensuring that the number of vertices in G and H is linearly bounded by the number of variables and clauses in and instance of 1-IN-3-SAT.

We now proceed with the proof of Theorem 3.

Proof. Let ψ be an instance of 1-IN-3-SAT with variable set $\{x_1, x_2, \ldots, x_n\}$ and clause set $\{C_1, C_2, \ldots, C_m\}$. We construct a corresponding instance (G, H) of LABELED CONTRACTION such that ψ is satisfiable if and only if H is a labeled contraction of G.

We simultaneously define the graphs G and H as follows (See Figure 4 for an illustration.):

- *Global vertices:* Add two special vertices g_T and g_F to both G and H, connected by the edge (g_T, g_F). These represent the "true" and "false" assignments, respectively, for literals.
- *Variable gadget:* For each variable x_i, $i \in [n]$:
  - Add a vertex u_i to both G and H, representing variable x_i.
  - Add two vertices v_i and v'_i to G, corresponding to literals x_i and $\neg x_i$, respectively.
  - In G, connect u_i to both v_i and v'_i. Also connect both v_i and v'_i to g_T and g_F.
  - In H, add edges (u_i, g_T), (u_i, g_F), and retain (g_T, g_F).
- *Clause gadget:* For each clause $C_j = (\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$, let $\mathsf{litv}(\ell)$ denote v_i if $\ell = x_i$, and v'_i if $\ell = \neg x_i$.
  - Add four vertices $w_{j0}, w_{j1}, w_{j2}, w_{j3}$ to both G and H.
  - In H, connect w_{j0} to g_T and g_F.
  - In G, connect w_{j0} to $\mathsf{litv}(\ell_{j1})$, $\mathsf{litv}(\ell_{j2})$, and $\mathsf{litv}(\ell_{j3})$.
  - For each $k \in \{1, 2, 3\}$, add edge (w_{jk}, g_T) to both G and H, and edge (w_{jk}, g_F) only to H.
  - Finally, form a 6-cycle in G by connecting the three w_{jk}'s with the three $\mathsf{litv}(\ell_{jk})$'s in an alternating cycle such that no two w_{jk}'s nor any two $\mathsf{litv}(\ell_{jk})$'s are adjacent.

This completes the reduction. It is easy to see that it takes polynomial time, and both graphs G and H have degeneracy at most 3.

We now show the equivalence between satisfiability of ψ and labeled contractibility of G into H.

($\Rightarrow$) Suppose there exists a satisfying assignment $\alpha : \{x_1, \ldots, x_n\} \to \{\texttt{True}, \texttt{False}\}$ such that exactly one literal is true in each clause. Define the contraction sequence S as follows:

- For each x_i, if $\alpha(x_i) = \texttt{True}$, then add edges (v_i, g_T), (v'_i, g_F) to S; and if $\alpha(x_i) = \texttt{False}$, then add edges (v_i, g_F), (v'_i, g_T) to S.

This ensures the correct neighborhood for u_i in H, as it will have edges to both g_T and g_F.

Now consider any clause C_j. Since exactly one literal is true under α, exactly one $\mathsf{litv}(\ell_{jk})$ is contracted into g_T, and the remaining two are contracted into g_F. The contraction of the w_{jk}-triangle via the 6-cycle then ensures:

- The edge (w_{j0}, g_T) is introduced by the literal assigned true;
- The edge (w_{j0}, g_F) is introduced by the literals assigned false;
- All three edges $(g_F, w_{j1}), (g_F, w_{j2}), (g_F, w_{j3})$ are created.

Hence, the resulting graph after the contraction is precisely H.

($\Leftarrow$) Conversely, suppose H is a labeled contraction of G. We argue that this implies a satisfying assignment for ψ.

The only way to create both edges (u_i, g_T) and (u_i, g_F) in H is by contracting v_i and v'_i into g_T and g_F, respectively, or vice versa. This naturally defines a valid assignment:

$$\alpha(x_i) = \begin{cases} \texttt{True} & \text{if } v_i \text{ is contracted to } g_T, \\ \texttt{False} & \text{if } v'_i \text{ is contracted to } g_T. \end{cases}$$

Next, observe that to produce all three edges $(g_F, w_{j1}), (g_F, w_{j2}), (g_F, w_{j3})$, at least two of the literals in clause C_j must contract to g_F, while one contracts to g_T. Thus, in each clause, exactly one literal must be set to true under α. Therefore, α satisfies ψ as a 1-IN-3-SAT assignment.

Each vertex in the variable and clause gadgets has degree at most 3 in G and at most 2 in H, so the degeneracy of both graphs is bounded by a constant. The size of G is linear in $n + m$, and the construction can be completed in polynomial time. Hence, if Labeled Contraction can be solved in time $2^{o(|V(G)|+|E(G)|)}$, then 1-IN-3-SAT can be solved in time $2^{o(n+m)}$, which contradicts ETH according to Proposition 21. This completes the proof of Theorem 3. ◀

5.2 Algorithmic Result

We first present proof of Theorem 4 establishing an alternate FPT algorithm and later mention its consequences. Recall that the theorem states that the Labeled Contraction problem admits an algorithm with running time $(\delta(H) + 1)^k \cdot |V(G)|^{\mathcal{O}(1)}$, where $k = |V(G) \setminus V(H)|$.

Let (G, H) be an instance of the Labeled Contraction problem. With sanity check, we can assume that $V(H) \subseteq V(G)$.

The goal is to decide whether there exists a valid witness structure $\mathcal{W} = \{W_h \subseteq V(G) \mid h \in V(H)\}$ such that contracting each W_h into the vertex h yields the graph H.

Let $k = |V(G) \setminus V(H)|$ denote the number of vertices in G that are not in H, and hence must be contracted. Each such vertex $x \in V(G) \setminus V(H)$ must be assigned to exactly one vertex $h \in V(H)$, corresponding to the witness set W_h into which x will be contracted.

Define a mapping $\phi : V(G) \to V(H)$ as follows: (*i*) For each $v \in V(H)$, set $\phi(v) = v$; (*ii*) For each $x \in V(G) \setminus V(H)$, $\phi(x)$ denotes the vertex in $V(H)$ to which x is assigned (i.e., the representative vertex of the witness set containing x). A mapping ϕ defines a candidate witness structure, and we wish to determine whether it corresponds to a valid contraction of G into H.

To bound the number of candidate functions ϕ, we exploit structural properties of the graph H. Let $\delta(H)$ denote the degeneracy of H. By definition, H admits an ordering $v_1, v_2, \ldots, v_n$ such that each vertex has at most $\delta(H)$ neighbors among its predecessors. This implies that H is $(\delta(H)+1)$-colorable, and such a coloring can be computed in polynomial time.

Fix a proper coloring $c : V(H) \to [\delta(H)+1]$. For each vertex $x \in V(G) \setminus V(H)$, let $N_H(x) := \{v \in V(H) \mid \{x, v\} \in E(G)\}$. If two vertices $y_1, y_2 \in N_H(x)$ belong to the same color class (i.e., $c(y_1) = c(y_2)$) and $\{y_1, y_2\} \notin E(H)$, then assigning x to either y_1 or y_2 would result in the edge $\{y_1, y_2\}$ being introduced during the contraction process – contradicting the assumption that H is the target graph. Therefore, for each $x \in V(G) \setminus V(H)$, there are at most $\delta(H)+1$ valid choices for $\phi(x)$. These choices corresponds at most one vertex in each color class. Thus, the total number of candidate assignments ϕ is bounded by $(\delta(H)+1)^k$. For each such assignment, we can verify in polynomial time whether the induced contraction yields H, by simulating the contraction and checking edge and label consistency. This yields an algorithm with running time $(\delta(H)+1)^k \cdot |V(G)|^{\mathcal{O}(1)}$. This completes the proof for Theorem 4

6 Brute-force Algorithm and its Optimality

We now prove Theorem 5. We establish a conditional lower bound showing that the brute-force algorithm described in the full version is essentially optimal, unless the Exponential Time Hypothesis (ETH) fails. For the full details of the brute-force algorithm and its exact running time analysis, we refer the reader to the longer version of this paper. We present the conditional lower bound proof here.

Our lower bound is derived via a polynomial-time reduction from the following problem, introduced by Fomin et al. [28], which has served as a hardness source for several contraction-based graph problems.

Cross Matching
Input: A graph G' together with a partition (A, B) of $V(G')$ such that $|A| = |B|$.
Question: Does there exist a perfect matching $M \subseteq E(G')$ such that each edge in M connects one vertex from A and one from B, and the graph G'/M is a clique?

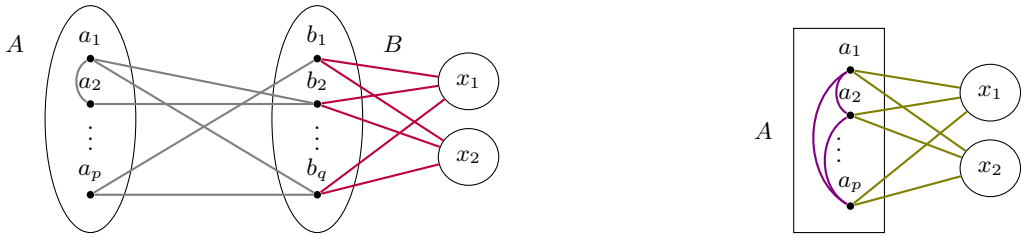
▶ **Proposition 22** (Lemma 4.1 in [28]). *Unless the ETH fails, Cross Matching cannot be solved in time $n^{o(n)}$, where $n = |A| = |B|$.*

We now present a polynomial-time reduction from Cross Matching to Labeled Contractibility, preserving the ETH-based lower bound under size-preserving transformation.

▶ **Lemma 23.** *Unless the ETH fails, the Labelled Contractibility problem does not admit an algorithm running in time $|V(H)|^{o(|V(G)|)}$.*

Proof. Let (G', A, B) be an instance of Cross Matching with $|A| = |B| = n$. We construct an equivalent instance (G, H) of Labeled Contractibility as follows.

The graph G is obtained from G' by introducing two new vertices x_1 and x_2, and adding edges between each of these new vertices and all vertices in B. Formally, $V(G) = V(G') \cup \{x_1, x_2\}$, and $E(G) = E(G') \cup \{(x_1, b), (x_2, b) \mid b \in B\}$. The target graph H is defined on vertex set $V(H) = A \cup \{x_1, x_2\}$, with the following edge set: (i) all pairs $\{a_1, a_2\} \subseteq A$ are adjacent (i.e., A induces a clique); and (ii) both x_1 and x_2 are adjacent to every vertex in A. We remark that vertices x_1 and x_2 are *not* adjacent. Equivalently, H is a complete graph on $A \cup \{x_1, x_2\}$ minus the edge (x_1, x_2). See Figure 5 for an illustration.



(a) Graph G. **(b)** Graph H.

Figure 5 Illustration of the graphs G and H in the reduction. The required adjacency across $\{x_1, x_2\}$ and A forces a candidate solution to form a matching across A and B.

We prove that (G', A, B) is a Yes-instance of Cross Matching if and only if (G, H) is a Yes-instance of Labeled Contractibility.

($\Rightarrow$) Suppose (G', A, B) is a Yes-instance of Cross Matching. Then there exists a perfect matching M between A and B such that the contracted graph G'/M is a clique.

We construct a labeled contraction sequence S on G by contracting each edge $(a, b) \in M$, with $a \in A$ and $b \in B$, in arbitrary order. Since M is a matching, all contracted edges are vertex-disjoint. After performing all contractions in S, each vertex $b \in B$ is removed, and the vertex set of G/S becomes $A \cup \{x_1, x_2\}$.

In G/S, the subgraph induced by A is a clique, since G'/M is a clique and the contractions only merged b into a. Furthermore, since x_1 and x_2 were connected to every $b \in B$, and each b was contracted into some $a \in A$, the contraction introduces edges from both x_1 and x_2 to every $a \in A$. However, since x_1 and x_2 are not adjacent to each other in the original graph G, and no contraction involved either of them, the edge (x_1, x_2) is not introduced. Thus, the resulting graph G/S is H.

($\Leftarrow$) Suppose (G, H) is a Yes-instance of Labelled Contractibility, i.e., there exists a sequence S of labeled contractions such that $G/S = H$.

By construction, $V(G) = A \cup B \cup \{x_1, x_2\}$ and $V(H) = A \cup \{x_1, x_2\}$. Therefore, all vertices in B must be removed during the contraction sequence. In the labeled contraction model, a vertex is removed only when it is the second vertex in a contracted pair. Hence, each $b \in B$ must appear as the second vertex in some contraction $(u, b) \in S$.

We now argue that each $b \in B$ must be contracted into a distinct vertex $a \in A$. Suppose, for contradiction, that some $a \in A$ does not serve as the representative of any contracted pair. Consider the edge (a, x_1) in H. Since a and x_1 were not adjacent in G, this edge must have been created through a contraction involving a common neighbor $b \in B$. The only way to generate such an edge is to contract b into x_1, i.e., perform $(x_1, b) \in S$. However, b is also adjacent to x_2 in G, and contracting b into x_1 would create an edge (x_1, x_2), a contradiction, since $(x_1, x_2) \notin E(H)$. Therefore, for each $a \in A$, there must be a distinct $b \in B$ such that $(a, b) \in S$. Since $|A| = |B| = n$, this yields a perfect matching $M = \{(a, b) \in S\}$ between A and B.

Finally, we argue that G'/M is a clique. Since $G/S = H$, and the subgraph induced by A in H is a clique, the same must hold in G'/M (as contractions outside $A \cup B$ do not affect this subgraph). Thus, (G', A, B) is a Yes-instance of Cross Matching.

The construction of (G, H) from (G', A, B) takes polynomial time. The sizes satisfy $|V(G)| = 2n+2$ and $|V(H)| = n+2$, both in $\mathcal{O}(n)$. Therefore, if Labeled Contractibility admitted an algorithm running in time $|V(H)|^{o(|V(G)|)}$, then Cross Matching could be solved in time $n^{o(n)}$, contradicting the ETH according to Proposition 22. This concludes the proof of the lemma. ◀

References

1 Akanksha Agarwal, Saket Saurabh, and Prafullkumar Tale. On the parameterized complexity of contraction to generalization of trees. *Theory of Computing Systems*, 63(3):587–614, 2019. doi:10.1007/s00224-018-9892-z.

2 Akanksha Agrawal, Lawqueen Kanesh, Saket Saurabh, and Prafullkumar Tale. Paths to trees and cacti. *Theoretical Computer Science*, 860:98–116, 2021. doi:10.1016/j.tcs.2021.01.033.

3 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. *ACM Transactions on Computation Theory*, 11(3):1–22, 2019. doi:10.1145/3319909.

4 Takao Asano and Tomio Hirata. Edge-contraction problems. *J. Comput. Syst. Sci.*, 26(2):197–208, 1983. doi:10.1016/0022-0000(83)90012-0.

5 Rémy Belmonte, Petr A. Golovach, Pim Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014. doi:10.1007/S00236-014-0204-Z.

6 Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. Tight lower bounds for problems parameterized by rank-width. *arXiv preprint*, 2210(02117), 2022. doi:10.48550/arXiv.2210.02117.

7 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for proper interval completion. *SIAM J. Discret. Math.*, 29(4):1961–1987, 2015. doi:10.1137/140988565.

8 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. Subexponential parameterized algorithm for interval completion. *ACM Trans. Algorithms*, 14(3):35:1–35:62, 2018. doi:10.1145/3186896.

9 Andries E. Brouwer and Henk Jan Veldman. Contractibility and np-completeness. *J. Graph Theory*, 11(1):71–79, 1987. doi:10.1002/jgt.3190110111.

10 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *Proc. of the 8th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 8246 of *LNCS*, pages 97–109, 2013. doi:10.1007/978-3-319-03898-8_10.

11 Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 1096–1115. SIAM, 2016. doi:10.1137/1.9781611974331.ch77.

12 Yixin Cao. Unit interval editing is fixed-parameter tractable. *Inf. Comput.*, 253:109–126, 2017. doi:10.1016/j.ic.2017.01.008.

13 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, 2015. doi:10.1145/2629595.

14 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.

15 Dibyayan Chakraborty, Oscar Defrain, Florent Foucaud, Mathieu Mari, and Prafullkumar Tale. Parameterized complexity of isometric path partition: treewidth and diameter. *arXiv preprint*, 2508(05448), 2025. doi:10.48550/arXiv.2508.05448.

16 Dipayan Chakraborty, Florent Foucaud, Diptapriyo Majumdar, and Prafullkumar Tale. Tight (double) exponential bounds for identification problems: Locating-dominating set and test cover. *35th International Symposium on Algorithms and Computation (ISAAC 2024)*, 322:19:1–19:18, 2024. doi:10.4230/LIPIcs.ISAAC.2024.19.

17 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2020. arXiv:2001.06867.

18 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification, 2020. arXiv:2001.06867.

19 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.

20 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Inf. Comput.*, 256:62–82, 2017. doi:10.1016/J.IC.2017.04.009.

21 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. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

22 Andreas Darmann and Janosch Döcker. On a simple hard variant of not-all-equal 3-sat. *Theor. Comput. Sci.*, 815:147–152, 2020. doi:10.1016/J.TCS.2020.02.010.

23 Pål Grønås Drange, Markus Fanebust Dregi, Daniel Lokshtanov, and Blair D. Sullivan. On the threshold of intractability. *J. Comput. Syst. Sci.*, 124:1–25, 2022. doi:10.1016/j.jcss.2021.09.003.

24 Pål Grønås Drange, Fedor V. Fomin, Michal Pilipczuk, and Yngve Villanger. Exploring subexponential parameterized complexity of completion problems. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 25 of *LIPIcs*, pages 288–299, 2014. doi:10.4230/LIPIcs.STACS.2014.288.

25 Pål Grønås Drange and Michal Pilipczuk. A polynomial kernel for trivially perfect editing. *Algorithmica*, 80(12):3481–3524, 2018. doi:10.1007/s00453-017-0401-6.

26 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

27 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.

28 Fedor V. Fomin, Daniel Lokshtanov, Ivan Mihajlin, Saket Saurabh, and Meirav Zehavi. Computation of hadwiger number and related contraction problems: Tight lower bounds. *ACM Trans. Comput. Theory*, 13(2):10:1–10:25, 2021. doi:10.1145/3448639.

29 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Comput.*, 42(6):2197–2216, 2013. doi:10.1137/11085390X.

30 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Problems in NP can admit double-exponential lower bounds when parameterized by treewidth or vertex cover. In *51st International Colloquium on Automata, Languages, and Programming, ICALP*, volume 297 of *LIPIcs*, pages 66:1–66:19, 2024. doi:10.4230/LIPICS.ICALP.2024.66.

31 Florent Foucaud, Esther Galby, Liana Khazaliya, Shaohua Li, Fionn Mc Inerney, Roohani Sharma, and Prafullkumar Tale. Metric dimension and geodetic set parameterized by vertex cover. In *42nd International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 327 of *LIPIcs*, pages 33:1–33:20, 2025. doi:10.4230/LIPICS.STACS.2025.33.

32 Petr A. Golovach, Marcin Kaminski, Daniël Paulusma, and Dimitrios M Thilikos. Increasing the minimum degree of a graph by contractions. *Theoretical Computer Science*, 481:74–84, 2013. doi:10.1016/j.tcs.2013.02.030.

33 Petr A. Golovach, Pim van't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013. doi:10.1016/j.tcs.2012.12.041.

34 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for Bipartite Contraction. *Information Processing Letters*, 113(22):906–912, 2013. doi:10.1016/j.ipl.2013.09.004.

35 Pinar Heggernes, Pim van 't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014. doi:10.1007/s00453-012-9670-2.

36 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013. doi:10.1137/130907392.

37 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.

38 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

39 Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. *SIAM Journal on Computing*, 0(0):STOC23–155–STOC23–214, 2023. doi:10.1137/23M1595059.

40 R. Krithika, Pranabendu Misra, and Prafullkumar Tale. A single exponential-time FPT algorithm for cactus contraction. *Theor. Comput. Sci.*, 954:113803, 2023. doi:10.1016/J.TCS.2023.113803.

41 Manuel Lafond and Bertrand Marchand. The parameterized landscape of labeled graph contractions. In *(To appear) in Algorithms and Data Structures Symposium WADS 2025*, volume abs/2502.16096, 2025. doi:10.48550/arXiv.2502.16096.

42 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

43 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

44 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *Proc. of the 8th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 8246 of *LNCS*, pages 243–254, 2013. doi:10.1007/978-3-319-03898-8_21.

45 Bertrand Marchand, Nadia Tahiri, Olivier Tremblay Savard, and Manuel Lafond. Finding maximum common contractions between phylogenetic networks. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI*, volume 312 of *LIPIcs*, pages 16:1–16:24, 2024. doi:10.4230/LIPICS.WABI.2024.16.

46 Bertrand Marchand, Nadia Tahiri, Olivier Tremblay-Savard, and Manuel Lafond. Finding Maximum Common Contractions Between Phylogenetic Networks. In *WABI*, pages 16:1–16:24, 2024. doi:10.4230/LIPIcs.WABI.2024.16.

47 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and $2K_2$-partition. *Journal of Combinatorial Theory, Series B*, 111:17–37, 2015. doi:10.1016/j.jctb.2014.09.002.

48 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011. doi:10.1007/978-3-642-22993-0_47.

49 Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. *Inf. Comput.*, 281:104812, 2021. doi:10.1016/J.IC.2021.104812.

50 Saket Saurabh and Prafullkumar Tale. On the parameterized complexity of maximum degree contraction problem. *arXiv preprint arXiv:2009.11793*, 2020.

51 Saket Saurabh and Prafullkumar Tale. On the parameterized complexity of maximum degree contraction problem. In *Proc. of the 15th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 180 of *LIPIcs*, pages 26:1–26:16, 2020. doi:10.4230/LIPIcs.IPEC.2020.26.

52 Prafullkumar Tale. Double exponential lower bound for telephone broadcast, 2024. doi:10.48550/arXiv.2403.03501.

53 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discret. Appl. Math.*, 3(2):151–153, 1981. doi:10.1016/0166-218X(81)90039-1.

54 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978. doi:10.1145/800133.804355.

Fall-Through Semantics for Mitigating Timing-Based Side Channel Leaks

Aniket Mishra
Indian Institute of Technology Gandhinagar, India

Abhishek Bichhawat
Indian Institute of Technology Gandhinagar, India

Abstract

With the recent advent of exploits like Spectre and Meltdown, the mitigation of side-channel attacks has become an important concern for security researchers. In this paper, we focus on timing-based side channels introduced through conditional branching on secret information within programs. We introduce a language that allows a programmer to write conditionals branching on secrets within its syntax, but has a semantics that keeps execution time constant with respect to an adversary under an observationally equivalent memory. We differ from other approaches that use program analysis methods, opting instead to modify the operational semantics to enforce the necessary properties. We formalize the semantics for our language with timing leak mitigations in Rocq (previously, Coq) and prove that these semantics satisfy the property of timing-sensitive non-interference. Since our system describes a mitigation approach for timing leaks in a general high-level imperative language, we believe that our semantics can be used as a basis for compiler construction for other high-level imperative languages that seek to be safe from timing side channels.

2012 ACM Subject Classification Theory of computation → Operational semantics; Theory of computation → Logic and verification

Keywords and phrases Timing leaks, information flow control, runtime monitor, type system, side-channel attacks

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.44

Supplementary Material *Software*: `https://github.com/satiscugcat/timing-side-channels/` archived at `swh:1:dir:ea55fb0069b70b63d7fed91de597f07191be8ed5`

Funding This work is funded in part by the DST Inspire Faculty Fellowship and the SERB Startup Research Grant.

Acknowledgements We would like to thank the anonymous reviewers for their feedback on this work.

1 Introduction

Traditionally, cybersecurity has dealt with the analysis of overt channels, which allow an attacker to both send and receive information, bypassing standard security protocols and opening up exploits that manipulate a system's behaviour. A side channel, on the other hand, is one where there is no such direct communication; instead, information is leaked through side effects of a program's execution. These side effects usually depend on the specific implementation of the program (which can vary even if it meets the input-output specifications) or the properties of the machine that executes the program. For instance, the amount of power the program consumes, the cache accesses, or even the time taken by the program to execute can leak information about secrets in the program [24]. Thus, side channels can leak information without the victim ever observing any change of the program's behaviour themselves.


© Aniket Mishra and Abhishek Bichhawat;
licensed under Creative Commons License CC-BY 4.0



One of the ways by which timing side channels are introduced is via the usage of control flow constructs in a program, as different branches of execution within the program may correspond to a difference in the time taken in the execution of each of these branches. An adversary that is able to track these times can leverage this information to steal data related to the branching invariant.

Cache side channels, on the other hand, are introduced due to the time taken to access data from the memory/cache. If some value has already been cached due to some prior access, it is much faster to retrieve that value as compared to it being accessed from the memory store. This difference in the time to access data from the cache and the main memory has been exploited as a cache side channel. Speculative execution has been a prominent reason for these side channel leaks and has been heavily exploited in the Spectre attacks [15, 19, 25, 39].

Another approach to avoid timing and cache-timing leaks in programs is to write code that follows *constant-time programming* [22]. Within this approach, certain programming practices such as avoiding the usage of conditionals, employing bit manipulation, etc. are followed to mitigate these side channels. However, not only does this lead a loss of expressivity and clarity with less straightforward code, it has also been shown that due to the presence of compiler optimisations, it is difficult to write code that is truly constant-time [20]. Moreover, it also becomes difficult to verify if a piece of code is constant-time in dynamic environments like the Web where a lot of the code is only available at execution time.

In this work, we propose a novel dynamic approach to tackle timing side channel leaks by modifying the runtime semantics of how a program executes. We mitigate this problem by formalising the semantics of a programming language, which syntactically allows its programs to contain conditionals, but its evaluation semantics are such that the programs are executed in a way that does not lead to timing leaks via control flow constructs. By modifying the semantics of how a program executes, we ensure that any optimizations included by the compilers are accounted for in ensuring that the program does not leak information due to timing side channel.

We propose a fall-through semantics that accounts for the timing leaks through branching in programs on secret values without requiring any static program transformations and without conflicting with the intended semantics of the program. We describe these semantics for an imperative language to show its enforcement. We prove that our fall-through semantics prevent any information leaks due to explicit flows, implicit flows and through timing side-channels by showing that our semantics satisfies timing-sensitive non-interference. We also prove that our approach is semantics-preserving. In addition to this, we have formalised these semantics within the Rocq (Coq) theorem prover [38] and thus have machine checked proofs [28] for the relevant security properties.

2 Background

We provide a brief overview of basic concepts in runtime enforcement of information flow control (IFC) [21, 37]. In dynamic IFC, a security label is attached with every value, which is an element of a pre-determined lattice and is an upper bound on the security levels of all the values that have influenced the computation that led to the value.

Typically, all values are classified into security levels or labels and the policy is a lattice over these labels. Information is only allowed to flow up the lattice having a well-defined partial ordering ($\sqsubseteq$), join ($\sqcup$), and meet ($\sqcap$) operations. For illustration purposes, often the smallest nontrivial lattice $L \sqsubseteq H$ is used, which specifies that public (low, L) data must not be influenced by confidential (high, H) data.

Information flows can happen either explicitly, implicitly or indirectly through covert side-channels.

Explicit flows arise as a result of values being assigned to others. For instance, there is an explicit flow of information from both z and y to x in the statement $x = y + z$. To account for explicit flows, we update the label of the computed value (x in our example) with the least upper bound of the labels of the operands in the computation.

Implicit flows arise from control dependencies. For example, in the program l = *false; if (h)* l = *true;*, there is an implicit flow from h to the final value of l. To handle these flows, we can maintain a program-context label (*pc*-label), which is an upper bound on the labels of all the values that have influenced the control flow thus far. Thus, if the value in h has label H, then *pc* will be H within the if branch. To prevent leaks from secret contexts to public variables, we disallow assignments to a variable carrying a public value, in a secret context (also known as the no-sensitive-upgrade (NSU) check [7]). When assigning inside a branch, the final computed value has the label of the operands and the program context to indicate the level of secrecy. This is necessary to guarantee the standard security property of non-interference.

The above checks guarantee a variant of non-interference that does not account for leaks due to indirect side-channels like termination or timing. Timing side channels are introduced as a result of the implementation of a program, specifically through control flow. Control flow constructs like conditionals introduce timing side channels because different branches of a conditional correspond to different execution times for a program. As an example of how this may leak information, consider a simple use case of a program that checks an input against some password.

```
1    bool matchpwd(int* input, size_t n){
2        if (n != pwd_length) return false;
3        for (int i = 0; i < n; i++) {
4            if (input[i] != pwd[i]) return false;
5        }
6        return true;
7    }
```

The first branch returns immediately if the actual password length is different from the length of the input. If the length of the input is same as that of the password, it enters the loop requiring more time for processing as compared to the former case. The loop iterates depending on how many characters of the password match with the input. Based on the *time taken* for the program to return, an adversary can figure out whether the length of the input matches the password and if the different characters in the input match that of the password.

Further, information stored in the cache can affect the timing behavior of program execution as data not in cache takes more time for retrieval as compared to data already in the cache. If the data has been retrieved in a secret context, the presence of data in the cache reveals information about the value of the secret context.

To guarantee timing-sensitive non-interference, we need to show that if any run of a program starting from a specific memory store takes time T to execute, then all runs of the same program starting from an equivalent memory store (that looks the same to an adversary who cannot observe secret values) take the same time T to execute while resulting in an equivalent final store. We also need to show that the order and timing of memory reads/writes remains the same across such equivalent memory stores.

2.1 Threat Model

An adversary may observe all public values in the memory store. More generally, an adversary at a security level ℓ in the lattice can observe all values in the memory store that are labelled $k \sqsubseteq \ell$; all such values are considered public with respect to the adversary while values that have labels $k \not\sqsubseteq \ell$ are secret to the adversary. Our threat model assumes that the adversary has knowledge of the address of every memory access and the point in time during the execution of the program where this was performed. The adversary can additionally observe the time it takes for a program to execute and whether or not a program has terminated. Finally, the adversary can also observe when any given memory location has been read from or written to during the execution of the program. We assume that the time needed to perform an operation is standard and does not depend on the number of bits set to 0/1 in the operands. Similarly, we assume that the time taken to read/write a value from/to a memory location is the same regardless of the value. However, our approach can easily be extended to account for these differently by storing fine-grained timing information for different constants, variables and their types.

3 Related Works

Prior approaches have focused on verification of absence of timing leaks in programs or generating constant-time code. Our approach, on the other hand, focuses on runtime enforcement of mitigation of timing related leaks in programs, which, to the best of our knowledge, has not been explored before. The major difference between our work and the prior works is that our focus is on building a runtime that can mitigate timing leaks as opposed to performing program transformations at the source-level. The idea is to enforce this at the architectural level to reduce additional sources of timing leaks. We discuss some of the aforementioned prior works next.

Johan Agat presented one of the first approaches to mitigating timing leaks by proposing a transformation of the program to account for time spent in each of the branches [1], specifically by inserting dummy code that would take the same amount of time to execute as the real code. The program counter security model [29] is another such work, which pursued the problem of timing side-channels by transforming C code using a static checker that detects violations due to timing channels. The paper also introduces a novel method to formally capture the behaviour of various kinds of side channels, call the "program counter transcript model". Future approaches built on top of these works or used their approach to provide more accurate and robust mitigation approaches by transforming original programs to constant-time code. While our work does not transform the program code, the runtime semantics emulates a similar behavior to prevent leaks through the timing channel.

Some approaches have been proposed to verify the absence of timing side-channel leaks in program source-codes [4, 8, 14, 31, 33, 34, 40]. Approaches have also targeted verifying programs at the intermediate code or representation level to guarantee freedom from timing leaks [2, 3, 9] while some other approaches provide the same guarantees for assembly or machine-code [14, 17]. Some testing and fuzzing-based approaches target generation of concrete inputs to test for the presence of timing vulnerabilities in code [5]. However, these approaches focus on detection of side-channel leaks rather than preventing leaks due to them.

Certain other works [13, 18, 26, 27, 32, 36, 39, 41, 42] use various programming language or software engineering-based approaches for program transformation or repair to modify a program to remove timing leaks. Borrello et al. [12] do this at the level of the LLVM IR. Many of these techniques also rely on random input testing/fuzzing, and thus do not

completely guarantee that the generated code is side channel free. All of these require the entire code to be available for analysis beforehand, and thus are unfit for dynamic runtime environments. Works such as those of Barthe et al. [10], Qin et al. [35], and Arranz Olmos et al. [6] present approaches to show the preservation of constant-time property of programs during compilation. Other works deal with providing guarantees by implementing DSLs which enforce the required properties at the language level, for example FaCT [16] for C, and SecWasm [11] for WASM. Swivel [30] and WaSCR [23] deal with the constant time execution of WASM code in its generality.

4 Language

4.1 Syntax

Next, we describe the syntax of a simple imperative language (shown in Figure 1) on which we describe our enforcement approach. Our approach focuses on modifying the runtime semantics of the language to ensure that program executions are free from any leaks due to timing side channels. Expressions and commands are represented by e and c, respectively. Binary operators are represented as $\oplus$. n represents a constant value (either a number, or a boolean) and x represents a variable, which stores a value n having the label k. The memory-store μ is a mapping from variable names (x) to labeled values, n^k. Commands include the standard no-op, assignment, sequencing, branching and looping commands while expressions include constants, variables and binary operations.

In our language, k represents a security label. The label pc represents the security level of the current program context.

$$\mu := \cdot \mid \mu, x \mapsto n^k$$
$$e := n \mid x \mid e_1 \oplus e_2$$
$$c := \mathsf{skip} \mid x := e \mid c_1; c_2 \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ e\ \mathsf{do}\ c$$

Figure 1 Syntax of WHILE.

Additionally, our semantics employ a 3-tuple structure of the form $\mathsf{debranch}(c, n, k)$, where n is a boolean and k is a level. This is not exposed in the surface level syntax, and is only used in intermediate evaluation steps.

4.2 Semantics

To formally define the information released by a program starting from some initial memory containing secret values, we define a big-step semantics for the language shown in Figure 1. To handle timing side channels, we introduce a debranch struct that accounts for timing leaks arising from the control flow of a program. Program configurations for expressions and commands are given as $e \mid \mu$ and $c \mid \mu$, respectively; the expression e and command c are respectively evaluated under the initial memory store μ, which is a map from variable names to labeled-values.

The judgement $e \mid \mu \Downarrow_{pc} T, n^k$ defines expression evaluation. It means that the expression e evaluated under the memory store μ in the program security context pc results in a constant n having the label k.

$$\text{CONST}\quad \frac{}{n \mid \mu \Downarrow_{pc} \langle \texttt{CONST} \rangle,\ n^{pc}} \qquad \text{VAR}\quad \frac{\mu(x) = n^k}{x \mid \mu \Downarrow_{pc} \langle \texttt{VAR}_\texttt{x} \rangle,\ n^{pc \sqcup k}}$$

$$\text{OPER}\quad \frac{e_1 \mid \mu \Downarrow_{pc} T_1, n_1^{k_1} \qquad e_2 \mid \mu \Downarrow_{pc} T_2, n_2^{k_2} \qquad n = n_1 \oplus n_2}{e_1 \oplus e_2 \mid \mu \Downarrow_{pc} \langle T_1, T_2, \texttt{OPER}_\oplus \rangle,\ n^{k_1 \sqcup k_2}}$$

Figure 2 Semantics for expression evaluation.

Command execution judgements have the form: $c \mid \mu \Downarrow_{pc} T \mid \mu'$ – where a command c executed under the memory store μ in the program security context pc modifies the initial store and generates a new memory store μ'.

Both judgements additionally generate T, which is a list of symbols corresponding to the evaluation steps taken by the program during its execution. For example $\langle \texttt{VAR}_\texttt{x}, \texttt{ASSN}_y \rangle$ would be list generated during evaluation of the command $y := x$. The notation T_1, T_2 represents the concatenation of the lists T_1 and T_2. The , operator indicates an append operation between two T lists. We assume that there is some constant extra amount of time that is required during each evaluation step of the program (where this constant may depend on which variable is being accessed, if any variable is accessed in the evaluation step). Under this assumption, proving the identity of the lists (as defined above) across execution traces over observationally equivalent memory stores proves our desired property, that is, the timing of each individual memory access is the same across observationally equivalent memory stores.

In any context where a boolean is expected (expressions involving the && and ! operators, within a conditional, or as the second element in the debranch 3-tuple), numbers are treated as falsy. This does not affect any of our results however, and any coercion scheme from numbers to booleans could be used.

The semantics for expression evaluation are defined in Figure 2 and the command execution semantics are defined in Figure 3. We describe the semantics in detail next.

The expression evaluation rules are mostly standard except for the additional timing related information captured by the semantics. The information captured is an account of the computation steps involved in the evaluation of the expression, that is dependent on the order of evaluation of the expressions. For instance, in CONST, $\langle \texttt{CONST} \rangle$ indicates that one constant expression has been evaluated; the resulting constant n has the label of the current program context pc. Similarly, for VAR, the variable x, evaluates to the value stored in μ while its security level is the join of the previous label k and the current program context pc, and generating the list $\langle \texttt{VAR}_\texttt{x} \rangle$, recording the variable that was read from too. The OPER inductively evaluates the sub-expressions e_1 and e_2 and the final T list is an order-dependent composition (thus, the expression $e_2 \oplus e_1$ would generate a different list) of the individual timing lists, such that each computation rule that is used in the evaluation of the final expression is recorded.

The SKIP command does not alter the memory store and returns a new list with SKIP while the sequencing command (SEQ) executes c_1 under the memory store μ and c_2 executes under the modified memory store μ'. The timing information of both these commands is added along with SEQ to record that a sequencing operation was executed. For the assignment statements (ASSN), we require that the final value stored in the variable has a label equivalent to at least the program context (which is obtained through the expression evaluation). The timing list generated also records the variable that it wrote to.

$$\text{SKIP}\quad \frac{}{\mathsf{skip} \mid \mu \Downarrow_{pc} \langle \texttt{SKIP} \rangle \mid \mu} \qquad \text{ASSN}\quad \frac{e \mid \mu \Downarrow_{pc} T, n^k}{x := e \mid \mu \Downarrow_{pc} \langle T, \texttt{ASSN}_{\texttt{x}} \rangle \mid \mu, x \mapsto n^k}$$

$$\text{SEQ}\quad \frac{c_1 \mid \mu \Downarrow_{pc} T_1 \mid \mu' \qquad c_2 \mid \mu' \Downarrow_{pc} T_2 \mid \mu''}{c_1; c_2 \mid \mu \Downarrow_{pc} \langle T_1, T_2, \texttt{SEQ} \rangle \mid \mu''}$$

$$\text{IF-HIGH}\quad \frac{e \mid \mu \Downarrow_{pc} T, n^k \qquad \mathsf{debranch}(c_1, n, pc) \mid \mu \Downarrow_k T_1 \mid \mu' \qquad \mathsf{debranch}(c_2, !n, pc) \mid \mu' \Downarrow_k T_2 \mid \mu'' \qquad k \not\sqsubseteq pc}{\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mu \Downarrow_{pc} \langle T, T_1, T_2, \texttt{IF} - \texttt{HIGH} \rangle \mid \mu''}$$

$$\text{IF-LOW}\quad \frac{e \mid \mu \Downarrow_{pc} T_1, n^k \qquad c = \left\{ \begin{array}{ll} c_1 & \mathit{if}\ n_1 = \mathsf{true} \\ c_2 & \mathit{otherwise} \end{array} \right\} \qquad c \mid \mu \Downarrow_{pc} T_2 \mid \mu' \qquad k \sqsubseteq pc}{\mathsf{if}\ e_1\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mu \Downarrow_{pc} \langle T_1, T_2, \texttt{IF} - \texttt{LOW} \rangle \mid \mu'}$$

$$\text{WHILE-F}\quad \frac{e \mid \mu \Downarrow_{pc} T, \mathsf{false}^k}{\mathsf{while}\ e\ \mathsf{do}\ c \mid \mu \Downarrow_{pc} \langle T, \texttt{WHILE} - \texttt{EXIT} \rangle \mid \mu}$$

$$\text{WHILE-T}\quad \frac{e \Downarrow_{pc} T_1, \mathsf{true}^k \qquad c \mid \mu \Downarrow_{pc} T_2 \mid \mu' \qquad \mathsf{while}\ e\ \mathsf{do}\ c \mid \mu' \Downarrow_{pc} T_3 \mid \mu'' \qquad k \sqsubseteq pc}{\mathsf{while}\ e\ \mathsf{do}\ c \mid \mu \Downarrow_{pc} \langle T_1, T_2, T_3, \texttt{WHILE} - \texttt{CONT} \rangle \mid \mu''}$$

Figure 3 Semantics for command evaluation.

For looping statements (WHILE-T), we enforce that the program context label is at least as secret as the label of the invariant, i.e., $k \sqsubseteq pc$ to ensure that the program does not loop on secret values; if the invariant evaluates to false (WHILE-F), the execution terminates. This is a standard assumption for constant-time execution as the maximum number of iterations a looping statement makes over all possible runs cannot be accurately determined. One could establish upper bounds, but any such upper bounds will naturally end up being extremely conservative (going up to possibly, $2^31 - 1$), and thus extremely inefficient performance-wise. Works such as those of Borrello et al. [12], that do try to provide such upper bounds do so via heuristics, and do not provide formal guarantees about the upper bound. The same restriction does not apply to conditionals, since either branch of a conditional runs at most one time (mutually exclusive to one another). The timing information accounts for the commands inside the loop.

The most interesting command evaluation rule is that of the branching statement. When branching on a predicate having a label less secret than the current program context (IF-LOW), the execution proceeds normally as in the unmodified semantics. However, when branching on predicates having a secret label (IF-HIGH), the execution follows the *debranching semantics* under the current pc as shown in Figure 4, which we explain next.

4.3 Debranching Semantics

The debranching command is given by $\mathsf{debranch}(c, n, \ell)$ where c is the command from the secret then branch ($n = \mathit{true}$) or the else branch ($n = \mathit{false}$) and ℓ is the previous security level at which the branching occurred.

DEB-SKIP

$$\frac{}{\mathsf{debranch}(\mathsf{skip}, n, \ell) \mid \mu \Downarrow_{pc} \langle \mathtt{DEB-SKIP} \rangle \mid \mu}$$

DEB-ASSN-TRUE

$$\frac{e \mid \mu \Downarrow_{pc} T_1, n^k \qquad x \mid \mu \Downarrow_{pc} T_2, n_1^{k_1}}{\mathsf{debranch}(x := e, (true), \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, \mathtt{DEB-ASSN_x} \rangle \mid \mu, x \mapsto n^k}$$

DEB-ASSN-FALSE

$$\frac{e \mid \mu \Downarrow_{pc} T_1, n^k \qquad x \mid \mu \Downarrow_{pc} T_2, n_1^{k_1}}{\mathsf{debranch}(x := e, (false), \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, \mathtt{DEB-ASSN_x} \rangle \mid \mu, x \mapsto n_1^k}$$

DEB-SEQ

$$\frac{\mathsf{debranch}(c_1, n, \ell) \mid \mu \Downarrow_{pc} T_1 \mid \mu' \qquad \mathsf{debranch}(c_2, n, \ell) \mid \mu' \Downarrow_{pc} T_2 \mid \mu''}{\mathsf{debranch}(c_1; c_2, n, \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, \mathtt{DEB-SEQ} \rangle \mid \mu''}$$

DEB-IF-HIGH

$$\frac{\begin{array}{c} e_1 \mid \mu \Downarrow_{\ell} T_3, n_1^{k_\ell} \qquad e_1 \mid \mu \Downarrow_{pc} T_4, n_1^{k_{pc}} \qquad k_\ell \not\sqsubseteq \ell \qquad n_1 \& \& n = n' \qquad !n_1 \& \& n = n'' \\ \mathsf{debranch}(c_1, n', \ell) \mid \mu \Downarrow_{k_{pc}} T_1 \mid \mu' \qquad \mathsf{debranch}(c_2, n'', \ell) \mid \mu' \Downarrow_{k_{pc}} T_2 \mid \mu'' \end{array}}{\mathsf{debranch}(\text{ if } e_1 \text{ then } c_1 \text{ else } c_2, n, \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, T_3, T_4, \mathtt{DEB-IF-HIGH} \rangle \mid \mu''}$$

DEB-IF-LOW

$$\frac{c = \left\{ \begin{array}{ll} \mathsf{debranch}(c_1, n, \ell) & \textit{if } n_1 = \mathsf{true} \\ \mathsf{debranch}(c_2, n, \ell) & \textit{otherwise} \end{array} \right\} \qquad \begin{array}{c} e_1 \mid \mu \Downarrow_{\ell} T_1, n_1^k \\ c \mid \mu \Downarrow_{pc} T_2 \mid \mu' \end{array} \qquad k \sqsubseteq \ell}{\mathsf{debranch}(\mathsf{if}\ e_1\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, n, \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, \mathtt{DEB-IF-LOW} \rangle \mid \mu'}$$

DEB-WHILE-F

$$\frac{e \mid \mu \Downarrow_{\ell} T, \mathsf{false}^k}{\mathsf{debranch}(\mathsf{while}\ e\ \mathsf{do}\ c, n, \ell) \mid \mu \Downarrow_{pc} \langle T, \mathtt{DEB-WHILE-EXIT} \rangle \mid \mu}$$

DEB-WHILE-T

$$\frac{\begin{array}{c} e \Downarrow_{\ell} T_1, \mathsf{true}^k \\ \mathsf{debranch}(c, n, \ell) \Downarrow_{pc} T_2 \mid \mu' \qquad \mathsf{debranch}(\mathsf{while}\ e\ \mathsf{do}\ c, n, \ell) \mid \mu' \Downarrow_{pc} T_3 \mid \mu'' \end{array} \qquad k \sqsubseteq \ell}{\mathsf{debranch}(\mathsf{while}\ e\ \mathsf{do}\ c, n, \ell) \mid \mu \Downarrow_{pc} \langle T_1, T_2, T_3, \mathtt{DEB-WHILE-CONT} \rangle \mid \mu''}$$

Figure 4 Debranching semantics.

We distinguish the debranching commands' timing information from the normal commands that account for the fact that a secret branch has been encountered. The DEB-SKIP executes as SKIP without any modification to the store.

The DEB-ASSN-TRUE and DEB-ASSN-FALSE rules define the semantics of assignment operations in secret contexts. Depending on whether the assignment operation is in the "true" branch or the "false" branch, either the evaluated value of e is written in the memory store or the original value of x is written back to the memory store. Note that the label of the value in x may change due to the additional program context label being joined with the label of the value in x. The timing includes the time taken to evaluate x, e and the memory store write, which is captured by $\mathtt{DEB-ASSN_x}$. Note that here too we record the variable which we write to.

In DEB-ASSN-FALSE we upgrade the label of the value in the store even though we do not change the value itself. Without this upgrade, the value may leak information because in one branch of the program, the value (and label) would get updated while in the other branch it would not. Thus, the visibility of the value in the store would be secret dependent. In our semantics, the value becomes hidden regardless of the branch the program takes.

The DEB-SEQ rule inductively evaluates c_1 and c_2 under the debranching semantics, the result of which is the final store μ''. The timing accounts for the time taken in executing c_1 and c_2 and an indication that these were sequenced commands ($\texttt{DEB} - \texttt{SEQ}$).

The rule that branches on more secret values in a secret context requires both c_1 and c_2 to run in the more secret context. In DEB-IF-HIGH, the context switches are made because of the no-sensitive-upgrade check [7]. The results of the actions performed on the store are dependent on the prior value of n and the value obtained through evaluation of e_1. The DEB-WHILE-T rule requires that the context does not increase any further and the evaluation happens in the current context. This prevents any non-deterministic behavior due to the change in the predicate values.

4.4 Example

With the help of a simple example, we explain how the debranching semantics addresses the issue of timing-based side-channel leaks.

```
1    while (i < l)
2    do
3        if (input[i] != pwd[i])
4        then r := 1
5        else skip;
6        i := i + 1;
7    return r
```

In the example above, consider a two-point lattice $L \sqsubseteq H$ and two runs of the program where

$$\mu_1 = [\texttt{l} \mapsto 3^L; \texttt{i} \mapsto 0^L; \texttt{input} \mapsto [1,2,3]^L; \texttt{pwd} \mapsto [1,2,3]^H]$$

$$\mu_2 = [\texttt{l} \mapsto 3^L; \texttt{i} \mapsto 0^L; \texttt{input} \mapsto [3,2,3]^L; \texttt{pwd} \mapsto [1,2,3]^H]$$

and the initial $pc = L$.

In the first run with μ_1, the WHILE-T computes the value $n = true$ with time $\texttt{VAR}_\texttt{i}, \texttt{VAR}_\texttt{l}, \texttt{OPER}_<$ and executes the sequencing command which then executes the branch. The branch predicate evaluates to $\texttt{false}^\texttt{H}$ with time $\texttt{VAR}_{\texttt{input}[0]}, \texttt{VAR}_{\texttt{pwd}[0]}, \texttt{OPER}_{!=}$ as the first elements are the same and hence the debranch instruction is called with $\mathsf{debranch}(r := 1, false, L)$ under $pc = H$. The DEB-ASSN-FALSE is called, which evaluates r and constant 1 with times $\texttt{DEB} - \texttt{VAR}_\texttt{r}$ and $\texttt{DEB} - \texttt{CONST}$ respectively. The time to update the store with the original value of $r = 0$ is captured with $\texttt{DEB} - \texttt{ASSN}_\texttt{r}$. The next command $\mathsf{debranch}(skip, true, L)$ under $pc = H$ executes without any change to the memory and records $\texttt{DEB} - \texttt{SKIP}$. The second part of the sequencing command is the assignment, which computes the value $i + 1$ generating times $\texttt{CONST}$, $\texttt{VAR}_\texttt{i}$ and $\texttt{OPER}_+$ followed by $\texttt{ASSN}_\texttt{i}$ to update i. Similarly, the next two runs with μ_1 execute without effectively changing the value of μ_1 and returning the value of $r = 0$ meaning the comparison was successful generating the same sequence of timing information.

In the second run with μ_2, the WHILE-T computes the value $n = true$ with time $\texttt{VAR}_\texttt{i}, \texttt{VAR}_\texttt{l}, \texttt{OPER}_<$ and executes the sequencing command which then executes the branch. The branch predicate evaluates to $\texttt{true}^\texttt{H}$ with time $\texttt{VAR}_{\texttt{input}[0]}, \texttt{VAR}_{\texttt{pwd}[0]}, \texttt{OPER}_{!=}$ as the first elements are not the same and hence the debranch instruction is called with $\mathsf{debranch}(r := 1, true, L)$

under $pc = H$. The DEB-ASSN-TRUE is called, which evaluates r and constant 1 with times $\texttt{DEB} - \texttt{VAR}_\texttt{r}$ and $\texttt{DEB} - \texttt{CONST}$ respectively. The time to update the store with the value of $r = 1$ is captured with $\texttt{DEB} - \texttt{ASSN}_\texttt{r}$. The next command $\mathsf{debranch}(skip, false, L)$ under $pc = H$ executes without any change to the memory and records $\texttt{DEB} - \texttt{SKIP}$. The second part of the sequencing command is the assignment, which computes the value $i+1$ generating times $\texttt{CONST}$, $\texttt{VAR}_\texttt{i}$ and $\texttt{OPER}_+$ followed by $\texttt{ASSN}_\texttt{i}$ to update i. The next two runs with μ_2' execute without effectively changing the value of μ_2' and returning the value of $r = 1$ meaning the comparison failed. The timing for the remaining two runs is the same as the first run.

From the two runs, we observe that the final values of r^H are indistinguishable to an L-adversary and the timing information in both the runs is syntactically equal. In the normal run of the program, the assignment of $r = 1$ is executed only in one run, which takes a different execution time as compared to the skip operation. Furthermore, within our semantics, all the variables are read from and written to at the same time regardless of secret values. In a normal run, this is not the case, as r is written to in one run, but not the other. This demonstrates the ability of our semantics to mitigate cache side channels too.

5 Formalising Timing Leaks

The end-to-end security property usually established to ensure confidentiality is non-interference. Non-interference means that two runs of the same program starting from any two memory stores that are *observationally equivalent* for any adversary end with two memory stores that are also observationally equivalent for that adversary. For our observation model, where the adversary also records the time taken to complete execution, the timing information for all runs of the program should be observationally equivalent to the adversary.

Observational equivalence of values and stores is formalized using the relation $\sim_\ell$, where ℓ is the level of the attacker, and defined in Definition 1 and 2.

Definition 1 states that for an adversary at level ℓ, two values are equivalent if they are either public with respect to the adversary and have the same base value or both secret with respect to the adversary. Definition 2 states that two memory stores are equivalent with respect to the attacker if all values in the two stores are equivalent with respect to the attacker. The formalization of each of these equivalences in Rocq are shown immediately following the formal definitions. In the Rocq code, `Level` indicates the label, while `rel` is the partial-order relation between labels. `MemStore` is the memory store mapping variables to labeled-values.

▶ **Definition 1** (Value Observational Equivalence). *Two values $v_1 = n_1^{l_1}$ and $v_2 = n_2^{l_2}$ are observationally equivalent at level ℓ, written $v_1 \sim_\ell v_2$, iff either:*

1. *$n_1 = n_2$, $l_1 = l_2 \sqsubseteq \ell$ (or)*
2. *$l_1 \not\sqsubseteq \ell$ and $l_2 \not\sqsubseteq \ell$*

```
Inductive ValueObservationalEquivalent {rel: Level -> Level -> Type} {
    latticeProof: JoinSemilattice rel}: Primitive -> Level -> Level ->
    Primitive -> Level -> Type :=
| LowProof {n1 n2: Primitive} {l1 l2 l: Level} (nEq: n1 = n2) (lEq: l1 =
    l2) : ValueObservationalEquivalent n1 l1 l n2 l2
| HighProof (n1 n2: Primitive) {l1 l2 l: Level} (l1High: rel l1 l ->
    False) (l2High: rel l2 l -> False): ValueObservationalEquivalent n1
    l1 l n2 l2.
```

▶ **Definition 2** (Memory Store Observational Equivalence). *Two memory stores μ_1 and μ_2 are observationally equivalent at level ℓ, written as $\mu_1 \sim_\ell \mu_2$ iff $\forall x. \mu_1(x) \sim_\ell \mu_2(x)$*

```
Definition MemStoreObservationalEquivalent {rel: Level -> Level -> Type}
    {latticeProof: JoinSemilattice rel} (mu1: MemStore) (l: Level) (mu2:
    MemStore): Type :=
forall (x: Var), @ValueObservationalEquivalent rel latticeProof (fst (mu1
     x)) (snd (mu1 x)) l  (fst (mu2 x)) (snd (mu2 x)).
```

Using the observational equivalence, we define the timing equivalence for a program c with respect to an attacker at level ℓ in Definition 3.

▶ **Definition 3** (Timing Sensitive Non-interference). *A program c is said to not leak information via timing side channels, for any arbitrary level ℓ, if for any two memory stores μ_1, μ_2, $c \mid \mu_1 \Downarrow_\ell T_1 \mid \mu_1'$ and $c \mid \mu_2 \Downarrow_\ell T_2 \mid \mu_2'$, then $\mu_1 \sim_\ell \mu_2 \implies (T_1 = T_2 \ \wedge \ \mu_1' \sim_\ell \mu_2')$*

6 Soundness and Correctness

6.1 Soundness

To prove timing sensitive non-interference for our semantics, we establish a few helper lemmas to prove the final theorem. All of our formalization is mechanically proven in Rocq [28]. The theorem statements in Rocq are shown in the listings following the formal definitions.

We start by establishing the equivalence of expression evaluation under equivalent memory stores (Lemma 6). The lemma states that evaluating an expression under two observationally equivalent memory stores results in values $n_1^{k_1}$ and $n_2^{k_2}$ that are also observationally equivalent at level ℓ. We show the timing equality for expression evaluation separately in Lemma 9.

▶ **Lemma 4** (Lower Bound on Label from Expression Evaluation).

$$\forall e.\forall \mu.\forall \ell.e \mid \mu \Downarrow_\ell T, n^k \implies \ell \sqsubseteq k$$

▶ **Corollary 5** (Lowest Value for Expression Labels).

$$\forall e.\forall \mu.\forall \ell.e \mid \mu \Downarrow_\ell T, n^k \wedge k \sqsubseteq \ell \implies k = \ell$$

▶ **Lemma 6** (Expression Equivalence under Memory Store Equivalence). *If $\forall e.\forall \ell.\forall \mu_1.\forall \mu_2.$ $e \mid \mu_1 \Downarrow_\ell T_1, n_1^{k_1}$ $e \mid \mu_2 \Downarrow_\ell T_2, n_2^{k_2}$ then $n_1^{k_1} \sim_\ell n_2^{k_2}$*

Proof. By induction on the expression e. ◀

```
Lemma MemStoreEquivalenceImplExpressionEquivalence:
  forall {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type}
    {lattice: JoinSemilattice rel}
    {e: Expression} {mu1 mu2: MemStore} {l k1 k2: Level}
    {n1 n2: Primitive} {T1 T2: TimingList}
    (p1: @ExpressionBigStep binop_eval rel lattice e mu1 l T1 n1 k1)
    (p2: @ExpressionBigStep binop_eval rel lattice e mu2 l T2 n2 k2)
    (memEq: @MemStoreObservationalEquivalent rel latticeProof mu1 l mu2),
  @ValueObservationalEquivalent rel latticeProof n1 k1 l n2 k2.
```

Theorem 7 shows that our debranching semantics preserve memory store equivalence, i.e., if we start from equivalent memory stores and execute a command c under the debranching semantics, we end up with equivalent memory stores.

▶ **Theorem 7** (Debranching preserves Memory Store Equivalence). *For any program c, levels ℓ, pc_1, pc_2 such that $l \sqsubset pc_1$ and $l \sqsubset pc_2$, and boolean values n_1, n_2 $\forall \mu_1.\forall \mu_2.\mu_1 \sim_\ell \mu_2 \implies$ $\mu_1' \sim_\ell \mu_2'$ where $\mathsf{debranch}(c, n_1, \ell) \mid \mu_1 \Downarrow_{pc_1} T_1 \mid \mu_1'$ and $\mathsf{debranch}(c, n_2, \ell) \mid \mu_2 \Downarrow_{pc_2} T_2 \mid \mu_2'$*

Proof. By induction on structure of c. ◀

```
Theorem DebranchPreservesMemEq
        {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
        {rel: Level -> Level -> Type}
        {lattice: JoinSemilattice rel} :
  forall {c: Command} {n1 n2: bool} {mu1 mu2 mu1' mu2': MemStore}
    {l pc1 pc2: Level} {T1 T2: TimingList}
    (p1: @DebranchBigStep binop_eval rel lattice
                          (Debranch c n1 l) mu1 pc1 T1 mu1')
    (p2: @DebranchBigStep binop_eval rel lattice
                          (Debranch c n2 l) mu2 pc2 T2 mu2')
    (memEq: @MemStoreObservationalEquivalent rel lattice mu1 l mu2)
    (l_rel_pc1: rel l pc1) (l_not_pc1: l <> pc1)
    (l_rel_pc2: rel l pc2) (l_not_pc2: l <> pc2),
  @MemStoreObservationalEquivalent rel latticeProof mu1' l mu2'.
```

Finally, we show that command evaluation preserves memory store equivalence, i.e., starting from attacker-equivalent stores, running a program c under our semantics results in attacker-equivalent final stores.

▶ **Theorem 8** (Evaluation preserves Memory Store Equivalence). *For any program c, levels pc if $\forall \mu_1.\forall \mu_2.\ c \mid \mu_1 \Downarrow_{pc} T_1 \mid \mu_1', c \mid \mu_2 \Downarrow_{pc} T_2 \mid \mu_2'$, and $\mu_1 \sim_\ell \mu_2$ then $\mu_1' \sim_\ell \mu_2'$*

Proof. By induction on c and applying Theorem 7. ◀

```
Theorem CommandPreservesMemEq
    {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type}
    {latticeProof: JoinSemilattice rel} :
  forall  {c: Command} {mu1 mu2 mu1' mu2': MemStore}
    {pc: Level} {T1 T2: TimingList}
    (p1: @CommandBigStep binop_eval rel latticeProof c mu1 pc T1 mu1')
    (p2: @CommandBigStep binop_eval rel latticeProof c mu2 pc T2 mu2')
    (memEq: @MemStoreObservationalEquivalent rel latticeProof mu1 pc mu2)
    ,
  @MemStoreObservationalEquivalent rel latticeProof mu1' pc mu2'.
```

Further, we prove that with our semantics, the time-taken to evaluate an expression under two different stores, μ_1 and μ_2 is the same (Lemma 9).

▶ **Lemma 9** (Timing Security for Expressions). *If $\forall e.\forall \ell.\forall \mu_1.\forall \mu_2.e \mid \mu_1 \Downarrow_\ell T_1$ and $e \mid \mu_2 \Downarrow_\ell T_2$, then $T_1 = T_2$*

```
Lemma ExpressionTimSec
    {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type} {lattice: JoinSemilattice rel}:
  forall {e: Expression} {pc1 pc2 k1 k2: Level} {mu1 mu2: MemStore}
    {n1 n2: Primitive} {T1 T2: TimingList},
    @ExpressionBigStep binop_eval rel lattice e mu1 pc1 T1 n1 k1 ->
    @ExpressionBigStep binop_eval rel lattice e mu2 pc2 T2 n2 k2 ->
  (T1 = T2).
```

Using the above result, we prove that the time taken for executing a command under the debranching semantics, starting from any two attacker-equivalent memory stores, is the same.

▶ **Theorem 10** (Debranch Timing Semantics). *For any program c, levels ℓ, pc_1, pc_2 such that $l \sqsubset pc_1$, $l \sqsubset pc_2$, and boolean values n_1, n_2,*

$$\forall \mu_1.\forall \mu_2.\mu_1 \sim_\ell \mu_2 \implies T_1 = T_2,$$

where $\mathsf{debranch}(c, n_1, \ell) \mid \mu_1 \Downarrow_{pc_1} T_1 \mid \mu_1'$ *and* $\mathsf{debranch}(c, n_2, \ell) \mid \mu_2 \Downarrow_{pc_2} T_2 \mid \mu_2'$

$$\text{N-CONST}\ \frac{}{n \mid \eta \Downarrow n} \qquad \text{N-VAR}\ \frac{\eta(x) = n}{x \mid \eta \Downarrow n} \qquad \text{N-OPER}\ \frac{e_1 \mid \eta \Downarrow n_1 \qquad e_2 \mid \eta \Downarrow n_2 \qquad n = n_1 \oplus n_2}{e_1 \oplus e_2 \mid \eta \Downarrow n}$$

Figure 5 Normal semantics for expression evaluation.

```
Lemma DebranchTimSec
    {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type} {lattice: JoinSemilattice rel}:
  forall  {c: Command} {n1 n2: bool} {mu1 mu2 mu1' mu2': MemStore}
    {l pc1 pc2: Level} {T1 T2: TimingList}
    (p1: @DebranchBigStep binop_eval rel lattice (Debranch c n1 l)
                          mu1 pc1 T1 mu1')
    (p2: @DebranchBigStep binop_eval rel lattice (Debranch c n2 l)
                          mu2 pc2 T2 mu2')
    (memEq: @MemStoreObservationalEquivalent rel lattice mu1 l mu2)
    (l_rel_pc1: rel l pc1) (l_not_pc1: l <> pc1)
    (l_rel_pc2: rel l pc2) (l_not_pc2: l <> pc2),
  T1 = T2.
```

Using the above lemmas and theorems, we show that our semantics are timing-sensitive non-interferent according to the Definition 3.

▶ **Theorem 11** (Timing Sensitive Non-interference). *If*

$$\forall c.\forall \ell.\forall \mu_1.\forall \mu_2.\mu_1 \sim_\ell \mu_2 \text{ and } c \mid \mu_1 \Downarrow_\ell T_1 \mid \mu_1' \text{ and } c \mid \mu_2 \Downarrow_\ell T_2 \mid \mu_2'$$

then $T_1 = T_2$

```
Theorem CommandTimSec
    {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type} {lattice: JoinSemilattice rel} :
  forall {c: Command} {mu1 mu2 mu1' mu2': MemStore}
         {pc: Level} {T1 T2: TimingList}
    (p1: @CommandBigStep binop_eval rel lattice c mu1 pc T1 mu1')
    (p2: @CommandBigStep binop_eval rel lattice c mu2 pc T2 mu2')
    (memEq: @MemStoreObservationalEquivalent rel lattice mu1 pc mu2),
  T1 = T2.
```

6.2 Correctness

We also prove the correctness of our semantics, i.e., we prove that our debranching semantics does not conflict with the intended semantics of the program. To do this, first we formalise what the intended semantics looks like. The semantics are shown in Figure 5 and Figure 6.

For expressions, we use judgement of the form: $e \mid \eta \Downarrow n$ and for commands, the judgement is of the form: $c \mid \eta \Downarrow \eta'$ where e is an expression, c is a command and η is a map from variables to primitive values, and n is a primitive value.

We start by defining store projection, which is a view of the store with values having no labels.

▶ **Definition 12** (Store Projection). *Given a memory store μ its projection $\mu_\downarrow$ is defined as the function such that* $\forall x.\mu(x) = n^k \iff \mu_\downarrow(x) = n$

Given these semantics, we prove the correctness of expression evaluation, the debranching commands and command evaluation.

$$\text{N-SKIP}\ \frac{}{\mathsf{skip} \mid \eta \Downarrow \eta} \qquad \text{N-ASSN}\ \frac{e \mid \eta \Downarrow n}{x := e \mid \eta \Downarrow \eta, x \mapsto n} \qquad \text{N-SEQ}\ \frac{c_1 \mid \eta \Downarrow \eta' \qquad c_2 \mid \eta' \Downarrow \eta''}{c_1; c_2 \mid \eta \Downarrow \eta''}$$

$$\text{N-IF}\ \frac{e \mid \eta \Downarrow n \qquad c = \left\{ \begin{array}{ll} c_1 & \textit{if } n_1 = \mathsf{true} \\ c_2 & \textit{otherwise} \end{array} \right\} \qquad c \mid \eta \Downarrow \eta'}{\mathsf{if}\ e_1\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \eta \Downarrow \eta'} \qquad \text{N-WHILE-F}\ \frac{e \mid \eta \Downarrow \mathsf{false}}{\mathsf{while}\ e\ \mathsf{do}\ c \mid \eta \Downarrow \eta}$$

$$\text{N-WHILE-T}\ \frac{e \Downarrow \mathsf{true} \qquad c \mid \eta \Downarrow \eta' \qquad \mathsf{while}\ e\ \mathsf{do}\ c \mid \eta' \Downarrow \eta''}{\mathsf{while}\ e\ \mathsf{do}\ c \mid \eta \Downarrow \eta''}$$

Figure 6 Normal semantics for command evaluation.

▶ **Theorem 13** (Expression Soundness). *Given an expression e we have if $\forall \mu. \forall pc.\quad e \mid \mu \Downarrow_{pc} T, n_1^k$, then $e \mid \mu_\downarrow \Downarrow n_1$*

The expression correctness lemma states that if an expression evaluates to a value in our instrumented semantics, it also evaluates to the same value in the normal semantics.

▶ **Lemma 14** (Debranch False Ident). *Given a command c, and levels pc, ℓ we have $\forall \mu. \mu_\downarrow = \mu'_\downarrow$ where $\mathit{debranch}(c, \mathit{false}, \ell) \mid \mu \Downarrow_{pc} T \mid \mu'$*

▶ **Lemma 15** (Debranch Soundness). *Given a command c, and levels pc, ℓ we have $\forall \mu. \mu'_\downarrow = \eta$ where $\mathit{debranch}(c, \mathit{true}, \ell) \mid \mu \Downarrow_{pc} T \mid \mu'$ and $c \mid \mu_\downarrow \Downarrow_{pc} \eta$*

The final correctness theorem states that if a command evaluates under the instrumented semantics to some memory store μ', the command execution under the normal semantics produces a projected memory store $\eta = \mu'_\downarrow$.

▶ **Theorem 16** (Command Soundness). *Given a command c, and level pc, we have $\forall \mu. \mu'_\downarrow = \eta$ if $c \mid \mu \Downarrow_{pc} T \mid \mu'$ then $c \mid \mu_\downarrow \Downarrow_{pc} \eta$*

Further, we also show a (partial) completeness theorem that states that if a program, without any while loops, is executed under the normal semantics and generates a store η, under the instrumented semantics, the program generates a store μ such that $\mu \downarrow = \eta$. A stronger theorem would require proving the same with all while loops having secret predicates removed, but it is difficult to model secret predicates in the normal semantics.

As it is difficult to reason about the completeness of our semantics with respect to the normal semantics because of the loops, we limit our analysis to programs without while loops in them, as defined using the inductive predicate `NoWhile`.

```
Inductive NoWhile : Command -> Type :=
| NoWhileSkip : NoWhile SkipCommand
| NoWhileAssn: forall {x e}, NoWhile (AssnCommand x e)
| NoWhileSeq: forall {c1 c2}, NoWhile c1 -> NoWhile c2 ->
                NoWhile (SeqCommand c1 c2)
| NoWhileIf: forall {e c1 c2}, NoWhile c1 -> NoWhile c2 ->
                NoWhile (IfCommand e c1 c2).
```

The completeness theorem requires the completeness of expression evaluation and debranching, which are shown next.

▶ **Lemma 17** (Expression Evaluation Completeness). *Given an expression e, we have if* $\forall \mu. \forall pc. \quad e \mid \mu \Downarrow n_1$, *then* $\exists\ T\ k.e \mid \mu_\downarrow \Downarrow_{pc} T, n_1^k$

▶ **Lemma 18** (Debranching Completeness). *Given a command c, levels ℓ, pc and a store μ if* `NoWhile`(c) *and* $c \mid \mu_\downarrow \Downarrow_{pc} \eta$, *then* $\exists\ T\ \mu'.$*debranch*$(c, \mathit{true}, \ell) \mid \mu \Downarrow_{pc} T \mid \mu'$ *and* $\mu'_\downarrow = \eta$

Theorem 19 shows the final (partial) completeness theorem.

▶ **Theorem 19** (Relative Completeness). *Given a command c, level pc and a store μ if* `NoWhile`(c) *and* $c \mid \mu_\downarrow \Downarrow_{pc} \eta$, *then* $\exists\ T\ \mu'.c \mid \mu \Downarrow_{pc} T \mid \mu'$ *and* $\mu'_\downarrow = \eta$

```
Theorem CommandSystemCompleteness
    {binop_eval: BinOp -> Primitive -> Primitive -> Primitive}
    {rel: Level -> Level -> Type}
    {latticeProof: JoinSemilattice rel}:
  forall {c: Command} {mu: MemStore} {nu: NormalStore} {pc: Level},
    NoWhile c ->
    @NormalBigStep binop_eval c (StoreProjection mu) nu ->
    prod (EX (fun T => EX (fun mu' =>
            @CommandBigStep binop_eval rel latticeProof c mu pc T mu')))
            (StoreProjection mu' = nu).
```

7 Discussion

In our threat model, we assume the attacker has knowledge of every memory access. We model this by providing extra annotations in all the rules involving reading or writing from a variable, thus incorporating the address into the timing information. By proving that the timing information generated is independent of secrets, we thus prove that the memory accesses the program performs are independent of secrets. Thus, we believe our approach can be extended to handle cache side channels as well. However, it would involve a considerate formalisation, which we consider to be outside the scope of our paper.

Further, our focus here is on establishing a theoretical approach to solving this problem via a runtime monitor, which is generic enough to work at low-level. We leave the implementation of the semantics and performance comparisons on a real architecture to future work.

8 Conclusion and Future Work

We present a runtime approach to mitigating timing side-channel leaks in program during execution by ensuring that programs execute in constant time. By showing the independence of memory access patterns from secret values, we also provide strong guarantees against cache side channels. We formalize the semantics for our approach and prove that it satisfies timing-sensitive non-interference in Rocq theorem prover. We also show that our semantics are correct with respect to the normal execution of the program and do not produce wrong results.

In the future, we would like to extend this approach to a full-fledged programming language like LLVM IR to ensure the feasibility of the approach in the presence of constructs like return, continue, etc. that break control flow of the program. This would also allow us to evaluate the performance overheads incurred by the approach as compared to the normal execution and study the trade-offs. Another avenue of research is into the treatment of looping on high values, whether we could relax the restrictions on them and thus cover a wider assortment of programs with our semantics.

References

1 Johan Agat. Transforming out timing leaks. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 40–53, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/325694.325702.

2 José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-Assurance and High-Speed Cryptography. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1807–1823, Dallas Texas USA, October 2017. ACM. doi:10.1145/3133956.3134078.

3 José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 53–70, USA, 2016. USENIX Association. event-place: Austin, TX, USA. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida.

4 Basavesh Ammanaghatta Shivakumar, Gilles Barthe, Benjamin Grégoire, Vincent Laporte, and Swarn Priya. Enforcing fine-grained constant-time policies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 83–96, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3560689.

5 Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. Decomposition instead of self-composition for proving the absence of timing channels. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 362–375, Barcelona Spain, June 2017. ACM. doi:10.1145/3062341.3062378.

6 Santiago Arranz Olmos, Gilles Barthe, Lionel Blatter, Benjamin Grégoire, and Vincent Laporte. Preservation of speculative constant-time by compilation. *Proc. ACM Program. Lang.*, 9(POPL), January 2025. doi:10.1145/3704880.

7 Thomas H. Austin and Cormac Flanagan. Efficient purely-dynamic information flow analysis. In *Proc. ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, pages 113–124, 2009. doi:10.1145/1554339.1554353.

8 J. Bacelar Almeida, Manuel Barbosa, Jorge S. Pinto, and Bárbara Vieira. Formal verification of side-channel countermeasures using self-composition. *Science of Computer Programming*, 78(7):796–812, July 2013. doi:10.1016/j.scico.2011.10.008.

9 Gilles Barthe, Sunjay Cauligi, Benjamin Grégoire, Adrien Koutsos, Kevin Liao, Tiago Oliveira, Swarn Priya, Tamara Rezk, and Peter Schwabe. High-Assurance Cryptography in the Spectre Era. In *S&P 2021 - IEEE Symposium of Security and Privacy*, Virtual, France, May 2021. doi:10.1109/SP40001.2021.00046.

10 Gilles Barthe, Benjamin Grégoire, and Vincent Laporte. Secure compilation of side-channel countermeasures: The case of cryptographic "constant-time". In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 328–343, 2018. doi:10.1109/CSF.2018.00031.

11 Iulia Basys, Maximilian Algehed, Alexander Sjösten, and Andrei Sabelfeld. Secwasm: Information flow control for webassembly. In Gagandeep Singh and Caterina Urban, editors, *Static Analysis*, pages 74–103, Cham, 2022. Springer Nature Switzerland. doi:10.1007/978-3-031-22308-2_5.

12 Pietro Borrello, Daniele Cono D'Elia, Leonardo Querzoni, and Cristiano Giuffrida. Constantine: Automatic side-channel resistance using efficient control and data flow linearization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 715–733, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3484583.

13 Robert Brotzman, Shen Liu, Danfeng Zhang, Gang Tan, and Mahmut Kandemir. Casym: Cache aware symbolic execution for side channel detection and mitigation. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 505–521, 2019. doi:10.1109/SP.2019.00022.

14 Luwei Cai, Fu Song, and Taolue Chen. Towards efficient verification of constant-time cryptographic implementations. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024. doi:10.1145/3643772.

15 Sunjay Cauligi, Craig Disselkoen, Klaus v. Gleissenthall, Dean Tullsen, Deian Stefan, Tamara Rezk, and Gilles Barthe. Constant-time foundations for the new spectre era. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 913–926, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3385412.3385970.

16 Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. Fact: a dsl for timing-sensitive computation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 174–189, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3314221.3314605.

17 Jia Chen, Yu Feng, and Isil Dillig. Precise Detection of Side-Channel Vulnerabilities using Quantitative Cartesian Hoare Logic. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 875–890, Dallas Texas USA, October 2017. ACM. doi:10.1145/3133956.3134058.

18 Goran Doychev, Boris Köpf, Laurent Mauborgne, and Jan Reineke. Cacheaudit: A tool for the static analysis of cache side channels. *ACM Trans. Inf. Syst. Secur.*, 18(1), June 2015. doi:10.1145/2756550.

19 Xaver Fabian, Marco Patrignani, Marco Guarnieri, and Michael Backes. Do you even lift? strengthening compiler security guarantees against spectre attacks. *Proc. ACM Program. Lang.*, 9(POPL), January 2025. doi:10.1145/3704867.

20 Lukas Gerlach, Robert Pietsch, and Michael Schwarz. Do compilers break constant-time guarantees? In *Financial Cryptography and Data Security (FC)*, 2025.

21 Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proc. 1982 IEEE Symposium on Security and Privacy*, pages 11–20, 1982. doi:10.1109/SP.1982.10014.

22 Guidelines for mitigating timing side channels against cryptographic implementations. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html, 2022.

23 Liyan Huang, Junzhou He, Chao Wang, and Weihang Wang. Wascr: A webassembly instruction-timing side channel repairer. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pages 4562–4571, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3696410.3714693.

24 Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 104–113, Berlin, Heidelberg, 1996. Springer-Verlag. doi:10.1007/3-540-68697-5_9.

25 Matthew Kolosick, Basavesh Ammanaghatta Shivakumar, Sunjay Cauligi, Marco Patrignani, Marco Vassena, Ranjit Jhala, and Deian Stefan. Robust constant-time cryptography. *Proc. ACM Program. Lang.*, 9(PLDI), June 2025. doi:10.1145/3729310.

26 Rui Lima, João F. Ferreira, and Alexandra Mendes. Automatic repair of java code with timing side-channel vulnerabilities. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 1–8, 2021. doi:10.1109/ASEW52652.2021.00014.

27 Cong Ma, Dinghao Wu, Gang Tan, Mahmut Taylan Kandemir, and Danfeng Zhang. Quantifying and mitigating cache side channel leakage with differential set. *Proc. ACM Program. Lang.*, 7(OOPSLA2), October 2023. doi:10.1145/3622850.

28 Machine checked proofs. https://anonymous.4open.science/r/timing-side-channels-47BE, 2025.

29 David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model: automatic detection and removal of control-flow side channel attacks. In

Proceedings of the 8th International Conference on Information Security and Cryptology, ICISC'05, pages 156–168, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11734727_14.

30 Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean Tullsen, and Deian Stefan. Swivel: Hardening WebAssembly against spectre. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1433–1450. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/narayan.

31 Van Chan Ngo, Mario Dehesa-Azuara, Matthew Fredrikson, and Jan Hoffmann. Verifying and Synthesizing Constant-Resource Implementations with Types. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 710–728, San Jose, CA, May 2017. IEEE. doi:10.1109/SP.2017.53.

32 Brandon Paulsen, Chungha Sung, Peter A. H. Peterson, and Chao Wang. Debreach: mitigating compression side channels via static analysis and transformation. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE '19, pages 899–911. IEEE Press, 2020. doi:10.1109/ASE.2019.00088.

33 Swarn Priya. *Formally computer-verified protections against timing-based side-channel attacks*. Theses, Université Côte d'Azur, November 2023. URL: https://hal.science/tel-04331805.

34 Jonathan Protzenko, Jean-Karim Zinzindohoué, Aseem Rastogi, Tahina Ramananandro, Peng Wang, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cătălin Hriţcu, Karthikeyan Bhargavan, Cédric Fournet, and Nikhil Swamy. Verified low-level programming embedded in F*. *Proceedings of the ACM on Programming Languages*, 1(ICFP):1–29, August 2017. doi:10.1145/3110261.

35 Qi Qin, JulianAndres JiYang, Fu Song, Taolue Chen, and Xinyu Xing. Dejitleak: eliminating jit-induced timing side-channel leaks. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, pages 872–884, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3540250.3549150.

36 Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: closing digital side-channels through obfuscated execution. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 431–446, USA, 2015. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/rane.

37 Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

38 Rocq theorem prover. https://github.com/rocq-prover/rocq, 2025.

39 Haifeng Ruan, Yannic Noller, Saeid Tizpaz-Niari, Sudipta Chattopadhyay, and Abhik Roychoudhury. Timing side-channel mitigation via automated program repair. *ACM Trans. Softw. Eng. Methodol.*, 33(8), November 2024. doi:10.1145/3678169.

40 Basavesh Ammanaghatta Shivakumar, Gilles Barthe, Benjamin Gregoire, Vincent Laporte, Tiago Oliveira, Swarn Priya, Peter Schwabe, and Lucas Tabary-Maujean. Typing High-Speed Cryptography against Spectre v1 . In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1094–1111, Los Alamitos, CA, USA, May 2023. IEEE Computer Society. doi:10.1109/SP46215.2023.10179418.

41 Luigi Soares, Michael Canesche, and Fernando Magno Quintão Pereira. Side-channel elimination via partial control-flow linearization. *ACM Trans. Program. Lang. Syst.*, 45(2), June 2023. doi:10.1145/3594736.

42 Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. Eliminating timing side-channel leaks using program repair. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, pages 15–26, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3213846.3213851.

A Characterization of Spartan Graphs and New Lower Bounds for Eternal Vertex Cover

Neeldhara Misra
IIT Gandhinagar, India

Saraswati Girish Nanoti
Indian Institute of Science, Bangalore, India

Abstract

The eternal vertex cover game is played between an attacker and a defender on an undirected graph G. The defender identifies k vertices to position guards initially. The attacker, on their turn, attacks an edge e, and the defender must move a guard along e to defend the attack. The defender may move other guards as well, under the constraint that every guard moves at most once and to a neighboring vertex. The smallest number of guards required to defend attacks forever is called the eternal vertex cover number of G, denoted $\mathsf{evc}(G)$.

For any graph G, $\mathsf{evc}(G)$ is at least $\mathsf{mvc}(G)$ (the vertex cover number of G). A graph is Spartan if $\mathsf{evc}(G) = \mathsf{mvc}(G)$. It is known that a bipartite graph is Spartan if and only if every edge belongs to a perfect matching. We show that the only König graphs that are Spartan are the bipartite Spartan graphs. We also give new lower bounds for $\mathsf{evc}(G)$, generalizing a known lower bound based on cut vertices. We finally show a new matching-based characterization of all Spartan graphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Eternal Vertex Cover, Vertex Cover, König Graphs, Spartan Graphs, Matchings

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.45

Related Version *ArXiV*: `https://arxiv.org/abs/2504.06832`

Funding *Neeldhara Misra*: Supported by the SERB Early Career Researcher Grant ECR/2018/002967.
Saraswati Girish Nanoti: Supported by CSIR.

Acknowledgements This work was done when the second author was a student at IIT Gandhinagar. The second author thanks CSIR for the support.

1 Introduction

Recall that a subset of vertices S is called a *vertex cover* if, for every edge $e = \{u, v\}$, at least one of u or v belongs to S. The eternal vertex cover game, introduced by Klostermeyer and Mynhardt [4], is a turn-based two-player game played on a graph between players typically referred to as the attacker and the defender. The defender is tasked with placing guards on the vertices of the graph to protect against attacks on any of its edges, while the attacker selects edges to attack. The defender's goal is to ensure that every attack can be countered by moving a guard along the attacked edge, while optionally moving additional guards, under the constraint that guards move at most once and to a neighboring vertex. When the defender is able to defend attacks forever, the positions of the guards at every stage of the game form a vertex cover.


© Neeldhara Misra and Saraswati Girish Nanoti;
licensed under Creative Commons License CC-BY 4.0

The minimum number of guards required for the defender to successfully defend against an infinite sequence of attacks is known as the *eternal vertex cover number*, denoted as $\mathsf{evc}(G)$. We also use $\mathsf{mvc}(G)$ to denote the *minimum* vertex cover number of G, which is the size of a smallest vertex cover of G. It is well known that [4]:

$$\underbrace{\mathsf{mvc}(G) \leqslant \mathsf{evc}(G)}_{\text{The placement of guardsis a vertex cover.}} \quad \text{and} \quad \underbrace{\mathsf{evc}(G) \leqslant 2 \cdot \mathsf{mvc}(G)}_{\text{Guard both endpoints of a maximum matching.}}$$

A natural question is to identify the graphs for which the lower bound is tight, i.e., when $\mathsf{evc}(G) = \mathsf{mvc}(G)$. Such graphs are called *Spartan* – here, the defender can maintain a vertex cover with the minimum number of guards, making them an optimal solution in terms of resource usage. We are interested in the question of what such graphs look like from a structural perspective.

In [2], Babu et al. obtain a characterization for a class of graphs that includes chordal graphs and internally triangulated planar graphs. We briefly recall this result. Let the graph class $\mathcal{F}$ denote the class of all connected graphs G for which each minimum vertex cover of G that contains all the cut vertices of G induces a connected subgraph in G. (A cut vertex is a vertex whose removal disconnects the graph.) Let $G(V, E)$ be a graph that belongs to $\mathcal{F}$, with at least two vertices, and $X \subset V$ be the set of cut vertices of G. Then it is shown [2] that G is Spartan if and only if for every vertex $v \in V \backslash X$, there exists a minimum vertex cover S_v of G such that $X \cup \{v\} \subset S_v$. It is also known that bipartite graphs are Spartan if and only if every edge belongs to a perfect matching (essentially elementary) [6]. Our first result involves expanding the scope of this characterization to König graphs, which are graphs where the minimum vertex cover equals the maximum matching, and hence a natural generalization of bipartite graphs. We show that the only König graphs that are Spartan are those that are also bipartite and satisfy the conditions for being Spartan within bipartite graphs:

▶ **Theorem 1.** *A König-Egerváry graph G is Spartan if and only if it is bipartite and essentially elementary.*

We also develop a series of increasingly stronger pre-requisites – i.e., necessary conditions – for a graph to be Spartan. To begin with, note that for a graph G which has more than one vertex, if $\mathsf{evc}(G) = \mathsf{mvc}(G)$, then every vertex v of G must belong to a minimum-sized vertex cover S_v. Indeed, if not, then from any configuration, attacking an edge incident to v will result in the attacker winning. A result in [2] takes this further to show that if $\mathsf{evc}(G) = \mathsf{mvc}(G)$, then each vertex must belong to a minimum-sized vertex cover which contains all the cut vertices.

We generalize this idea further in the following way. Let S be a vertex cover of G and let $I = V(G) \setminus S$ be an independent set of G. A "bad situation" for the defender who has their guards positioned on the vertices of S is the following: there is a $T \subseteq I = V(G) \setminus S$ such that one of the connected components C of $G \setminus T$ is such that $|V(C) \cap S| = \mathsf{mvc}(C)$. This is because the attacker can attack an edge that has one endpoint in C and the other in T, forcing a guard out of C, and it is easy to check that the shortfall in $G[C]$ "cannot be fixed", creating a vulnerable edge that will lead to a win for the attacker (see Figure 1 where the defender has one guard on the vertex q with respect to the attack qb).

A vertex cover S is *weakly good* if there is no subset T of $V(G) \setminus S$ for which the bad situation described here occurs. We show that a graph G is Spartan only if every vertex is contained in a minimum-sized and weakly good vertex cover.

From the defender's perspective, sometimes it is not enough to just avoid the bad situation described above with the vertex covers that they work with. Indeed, suppose we have a component C as shown on the right side of Figure 1. Here, $|V(C) \cap S| > \mathsf{mvc}(C)$, but note

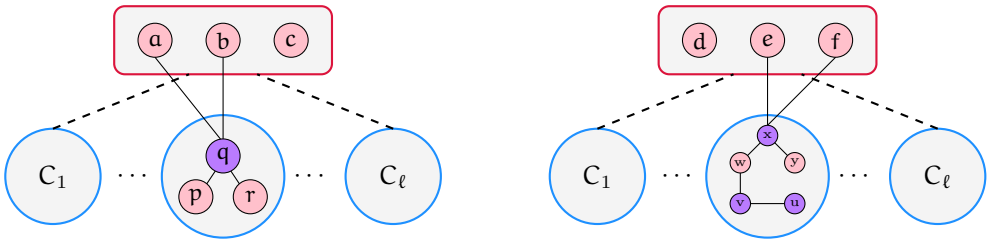


Figure 1 Examples of bad situations. Left: an example of a bad scenario with $T = \{a, b, c\}$. Note that the minimum vertex cover number of the component with the vertices $\{p, q, r\}$ is one and the size of its intersection with the vertex cover of G, i.e., $\{q\}$ is also one. Right: an example of a bad scenario with $T = \{d, e, f\}$. Here, the minimum vertex cover number of the component with the vertices $\{u, v, w, x, y\}$ is two and the intersection of this component with the vertex cover of G is $\{x, u, v\}$; however the set $\{u, v\}$ is not compatible with any minimum-sized vertex cover of the component.

that if the attacker attacks xe, then the defender is stuck with guards on v and u: while in principle $\mathsf{mvc}(C) = 2$, these two guards cannot be moved in a way that will defend all edges in the component C, and as before there is no pathway for guards from outside C to enter C.

While not as stark as before, this is a more nuanced bad scenario. Informally speaking, vertex covers that avoid even this scenario are called *strongly good* vertex covers. We show that a graph is Spartan only if every vertex is contained in a minimum-sized and strongly good vertex cover.

Our final and main result is a characterization of Spartan graphs at large. Let us begin by making a defender's strategy structurally explicit. To this end, consider a graph where we are able to find a non-empty family $\mathcal{F}$ of minimum-sized vertex covers such that the following holds:

For every $S \in \mathcal{F}$ and for every edge $uv \in G$ such that $u \in S$ and $v \notin S$, there exists $T \in \mathcal{F}$ such that $v \in T$ and T is reachable from S via guard movement while defending uv: this is to say that if the defender had guards positioned on S, then they will be able to move them in such a way that the guards finally end up on T and one of the guards moves along the edge uv.

Notice that such a family $\mathcal{F}$ naturally translates to a strategy for the defender: the defender can start with guards positioned on S for any $S \in \mathcal{F}$, and if any edge uv is attacked, the defender can either exchange guards (if both u and v are in S) or move guards to the T guaranteed by the property above. This would work indefinitely since every vertex cover in $\mathcal{F}$ offers the same opportunities to the defender.

This brings us to the question of capturing movements between minimum-sized vertex covers S and T. We refer to the vertices in $V \setminus (S \cup T)$ as the *dead zone*. The defender hopes to move guards currently positioned at S to a new target vertex cover T while also defending an arbitrary but fixed edge uv with (say) $u \in S \setminus T$ and $v \in T \setminus S$. Notice that the defender can transform the configuration of guards from S to T while defending uv if there are $|S \setminus T| - 1$ vertex-disjoint paths between $(S \setminus T) \setminus \{u\}$ and $(T \setminus S) \setminus \{v\}$ that do not use any vertices from the dead zone. We say that S and T are *mutually reachable* if such paths exist.

It is intuitive that the existence of a family of mutually reachable minimum vertex covers is a necessary and sufficient condition for a graph to be Spartan: indeed, given such a family, the defender has a strategy, and if the defender has a strategy, then the guard positions naturally correspond to such a family. Our main contribution is to capture the requirement of mutual reachability between vertex covers S and T in terms of matchings in an appropriately

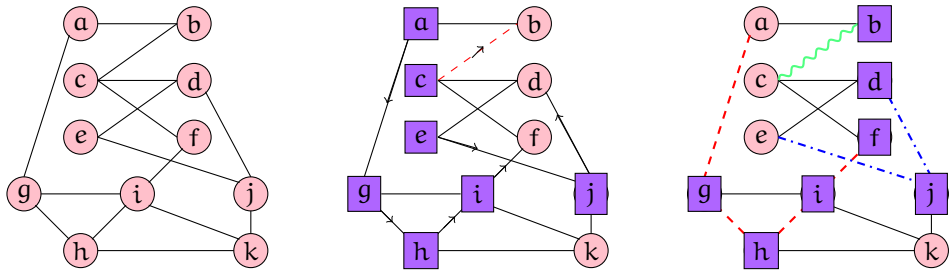


Figure 2 Demonstrating reachability between vertex covers $S = \{a, c, e, g, h, i, j\}$ and $T = \{b, d, f, g, h, i, j\}$ with the vertex $\{k\}$ in the dead zone. Here $S \setminus T = \{a, c, e\}$ and $T \setminus S = \{b, d, f\}$. There are three vertex-disjoint paths $P_1 = aghif$, $P_2 = ejd$ and $P_3 = cb$ (the attacked edge), from $S \setminus T$ to $T \setminus S$ with internal vertices from $S \cap T$. Thus, the guard on i moves to f, the guard on h moves to i, the guard on g moves to h and the guard on a moves to g. Notice that this movement happens along the path P_1. Similar movements happen along P_2 and P_3 and thus the guards reconfigure from S to T. Note that no vertex of P_1, P_2 and P_3 is common, if there was a common vertex then the guard which just arrived on the vertex could not have moved again before the next turn. Hence it is important that the paths P_1, P_2 and P_3 are vertex-disjoint when we want to use them to reconfigure from S to T.

defined auxiliary graph $\mathfrak{h}_G(S, T)$ based on S and T. The notion of the auxiliary graph required to translate the condition of mutual reachability in terms of matchings is somewhat technical and we omit the specifics from the introduction. The non-trivial technical aspect is to ensure that the reachability guaranteed by disjoint paths as described above is in fact captured by the existence of a matching in $\mathfrak{h}_G(S, T)$.

While it is not clear that our characterization is efficiently testable, we propose it with the goal of offering structural insight into the class of general Spartan graphs, and hope that this lays the groundwork for future work on algorithmic aspects of recognizing Spartan graphs.

2 Preliminaries

For any natural number $n \in \mathbb{N}$, the set $[1, n]$ denotes all the natural numbers between 1 and n (both included). Let $G = (V, E)$ be a simple, finite, undirected graph with $n > 1$ vertices and m edges (unless mentioned otherwise). Also unless mentioned otherwise, we will assume that the graph G is connected (and in particular, has no isolated vertices). We use the standard notation from [3]. If S is a subset of V, $G[S]$ denotes the subgraph of G induced by the vertices in S. The set of all vertices v such that $uv \in E(G)$ is denoted by $N(u)$ and is said to be the *open neighbourhood* of u. The set $N(u) \cup \{u\}$ is said to be the *closed neighbourhood* of u.

A *path* on n vertices is a graph P with vertices $\{v_1, v_2, \ldots, v_n\}$ such that each $(v_i, v_{i+1}) \in E(P)$ for $i \in [1, n-1]$. A *cycle* on n vertices is a graph C with vertices $\{v_1, v_2, \ldots, v_n\}$ such that each $(v_i, v_{i+1}) \in E(C)$ for $i \in [1, n-1]$ and $(v_n, v_1) \in E(C)$. A graph $G(V, E)$ is said to be *connected*, if for every pair of distinct vertices $u, v \in V$, there exists a path between u and v.

A subset S of $V(G)$ is said to be a *vertex cover* of G if for every edge $(u, v) \in E$, either $u \in S$ or $v \in S$ (or both). The size of a smallest vertex cover of the graph G is called the *minimum vertex cover number* of G and denoted by $\mathsf{mvc}(G)$. The graphs with $\mathsf{evc}(G) = \mathsf{mvc}(G)$ are known as *Spartan graphs*.

A subset S of $V(G)$ is said to be an *independent set* of G if no edge of $E(G)$ has both its endpoints in S. It can be seen that the complement of a vertex cover will be an independent set and vice versa.

A *matching* is a set of edges with no common endpoint. A *perfect matching* is a matching which contains one edge adjacent to each vertex of a graph. A graph may not always have a perfect matching. A matching of the largest cardinality in a graph is called a *maximum matching* and the size of this matching is denoted by $\mathfrak{mm}(G)$.

A *bipartite graph* $G = (V, E)$ is a graph whose vertex set can be partitioned into two independent sets, say $V = (A \cup B)$ such that every edge is between a vertex in A and one in B. Clearly, both A and B are vertex covers of G. If a bipartite graph $G = (A \cup B, E)$ is connected and its *only* minimum-sized vertex covers are A and B, then we say that G is *elementary*. If every connected component of a bipartite graph is elementary, then we call it *essentially elementary*.

In Section 4, we need to use the following result about bipartite graphs which is given in [5] and can also be obtained by a slight modification of a result in [6]. We give an explicit proof for completeness.

▶ **Lemma 2** ([6]). *Let G be a connected bipartite graph such that $V(G) = A \cup B$ where A and B are non-empty independent sets. If G has at least one perfect matching and G is non-elementary, then there exist non-empty $S \subsetneq A$ and $T \subsetneq B$ such that $|N(S)| = |S|$ and $|N(T)| = |T|$.*

Proof. Let G be a non-elementary connected bipartite graph with $V(G) = A \cup B$, where A and B are non-empty independent sets, and a perfect matching. Suppose that for every non-empty subset S of A, $S \subsetneq A$ we have $|N(S)| > |S|$. Let $ab \in E(G)$. We show that there exists a perfect matching between $A \setminus \{a\}$ and $B \setminus \{b\}$. Consider any non-empty $X \subseteq A \setminus \{a\}$. If $|N(X) \cap (B \setminus \{b\})| < |X|$, then $|N(X) \cap B| \leqslant |X|$. That is, $X \subsetneq A$ such that $|N(X)| \leqslant |X|$, which is contrary to our assumption that for every subset S of A, $S \subsetneq A$ we have $|N(S)| > |S|$. Thus, we cannot have any $X \subseteq A \setminus \{a\}$ such that $|N(X) \cap (B \setminus \{b\})| < |X|$. Therefore, by Hall's theorem, there exists a perfect matching M_1 between $A \setminus \{a\}$ and $B \setminus \{b\}$. Thus, $M = M_1 \cup \{ab\}$ is a perfect matching of G containing ab. Since ab was an arbitrary edge of G, every edge of G belongs to some perfect matching, and G is connected. This implies that G is elementary which is a contradiction. Therefore, there must exist a non-empty subset S of A such that $S \subsetneq A$ and $|N(S)| \leqslant |S|$ but $|N(S)|$ cannot be less than $|S|$ because G has a perfect matching. Thus $|N(S)| = |S|$. With a symmetric argument, we get that there exists a non-empty $T \subsetneq B$ such that $|N(T)| = |T|$. ◀

3 New Lower Bounds

In this section we give some necessary conditions which a graph G must satisfy, in order to have $\mathsf{evc}(G) = \mathsf{mvc}(G)$. One such condition is stated in [2] which is as follows:

▶ **Lemma 3** ([2]). *For a graph G which has more than one vertex, if $\mathsf{evc}(G) = \mathsf{mvc}(G)$ then every vertex v of G must belong to a minimum-sized vertex cover S_v.*

For understanding the nature of Spartan graphs, it is sufficient to only look at connected components. The lemma below which was proven in [6] justifies this.

▶ **Lemma 4** ([6]). *Let G be a Spartan graph with connected components $C_1, \ldots, C_\ell$. G is Spartan if and only if $G[C_i]$ is Spartan for all $1 \leqslant i \leqslant [\ell]$.*

Thus, we now look at only connected graphs.

▶ **Lemma 5.** *If a graph* G *with more than one vertex is Spartan and* I *is a maximum independent set of* G*, then there is a matching of* I *to* $V(G) \setminus I$*, which saturates* I*.*

Proof. Consider a Spartan graph G (with more than one vertex), and let I be a maximum independent set of G and $C = V(G) \setminus I$ be a minimum-sized vertex cover. Suppose there is no matching from I to S which saturates I, then by Hall's theorem, there exists $X \subseteq I$ such that $|N(X)| < |X|$. (Here, the neighborhood of any subset of I in the entire graph is the same as the neighborhood of that subset in C because I is an independent set.)

Consider an inclusion-wise minimal such subset X of I. Then for every $x \in X$, there exists a perfect matching from $X \setminus \{x\}$ to $N(X)$. If not, then there exists a $Y \subseteq X \setminus \{x\}$ such that $|N(Y)| < |Y|$ and as $Y \subseteq X \setminus \{x\}$ which means that $Y \subset X$, this contradicts the inclusion-wise minimality of X. Hence, there exists a perfect matching from $X \setminus \{x\}$ to $N(X)$ and this also means that $|X| = |N(X)| + 1$. Next, we make a claim which demonstrates how any vertex cover of G intersects the vertices in $X \cup N(X)$ and its proof is given in the Appendix.

▷ Claim 6. Any minimum-sized vertex cover of G cannot contain more than $|N(X)|$ vertices from $X \cup N(X)$.

Now consider a vertex $x \in X$. Since G is Spartan, there is a minimum-sized vertex cover S_x of G such that $x \in S_x$ by Lemma 3. We have already shown that $X \setminus \{x\}$ has a perfect matching with $N(X)$. Thus, any vertex cover must contain $|N(X)|$ vertices from $X \setminus \{x\} \cup N(X)$. Thus, S_x contains x and $|N(X)|$ vertices from $X \setminus \{x\} \cup N(X)$. That is, S_x contains $|N(X)|$ vertices from $X \cup N(X)$. This contradicts the above claim. Thus, the existence of a Hall's violator set X is not possible and there exists a matching from I to $S = V(G) \setminus I$ which saturates I. ◀

This immediately gives us a simple corollary (proof in Appendix).

▶ **Corollary 7.** *If* G *is a Spartan graph with more than one vertex and* $|V(G)| = n$*, then* $mvc(G) \geqslant n/2$*.*

▶ **Definition 8.** *A vertex cover* S *of a graph* G *is said to be a* weakly good *vertex cover if it satisfies the following: Let* $I = V(G) \setminus S$ *be the corresponding independent set. For every* $T \subseteq I$*, if* $C_1, C_2, \ldots C_\ell$ *are the connected components of* $G[V(G) \setminus T]$*, we must have* $|V(C_i) \cap S| > mvc(C_i)$ *(for all* $i \in [1, \ell]$*).*

When we are dealing with more than $mvc(G)$ guards, and we are in the model of the game where more than one guard is able to occupy a single vertex, then we will need to look at configurations of guards, not just vertex covers or vertex sets. A configuration of guards is a description of how many guards are present on each vertex of the graph G. A configuration can also be viewed as a function which maps each vertex to a non-negative integer such that the sum of the values of this function on all the vertices of G is equal to the number of guards. At some places, we will also view a configuration as a multi-set where a vertex appears as many times as the number of guards present on it. The manner in which we are using the word configuration, will be clear from context. When we say a configuration of guards $\mathcal{C}$ on a vertex cover S, we mean that the vertices having one or more guards in $\mathcal{C}$ (i.e., a non-zero value for the configuration function) are precisely the ones in S.

▶ **Definition 9.** *A configuration of guards* $\mathcal{C}$ *on a vertex cover* S *is said to be a* weakly good *configuration if it satisfies the following: Let* $I = V(G) \setminus S$ *be the corresponding independent set. For every* $T \subseteq I$*, if* $C_1, C_2, \ldots C_\ell$ *are the connected components of* $G[V(G) \setminus T]$*, the number of guards in* C_i *must be greater than* $mvc(C_i)$ *(for all* $i \in [1, \ell]$*).*

If the number of guards on G is the same as $\mathsf{mvc}(G)$, or we are working in the model where only one guard per vertex is allowed, then the vertex cover formed by vertices occupied by the guards in a weakly good configuration is a weakly good vertex cover. It is easier to see the notion of a weakly good vertex cover (or configuration) by looking at a vertex cover (or configuration) which is not weakly good.

▶ **Definition 10.** *Let S be a vertex cover of a graph G such that S is not weakly good. There exists a non-empty $T \subseteq I = V(G) \setminus S$ such that if $C_1, C_2, \ldots C_\ell$ are the connected components of $G[V(G) \setminus T]$, we must have $|V(C_i) \cap S| = \mathsf{mvc}(C_i)$ for some C_i (where $i \in [1, \ell]$). Then we will denote T as a* weakly bad set *corresponding to $\mathcal{C}$.*

▶ **Definition 11.** *Let S be a vertex cover of a graph G and let $\mathcal{C}$ be a configuration of guards on S such that $\mathcal{C}$ is not weakly good. There exists a non-empty $T \subseteq I = V(G) \setminus S$ such that if $C_1, C_2, \ldots C_\ell$ are the connected components of $G[V(G) \setminus T]$, there must exist a C_i (where $i \in [1, \ell]$) such that the number of guards in C_i must be equal to $\mathsf{mvc}(C_i)$ (cannot be less than $\mathsf{mvc}(C_i)$ because S is a vertex cover). Then we will denote T as a* weakly bad set *corresponding to S.*

For every vertex cover S which is not weakly good, there must exist at least one corresponding weakly bad set $T \subset V(G) \setminus S$. Similarly for every configuration $\mathcal{C}$ which is not weakly good, there must exist at least one corresponding weakly bad set $T \subset V(G) \setminus S$. Conversely, if for a vertex cover S (or a configuration $\mathcal{C}$ occupying the vertices in a vertex cover S), there exists a subset T of $V(G) \setminus S$ which is weakly bad, then the vertex cover S (or the configuration $\mathcal{C}$) is not weakly good.

▶ **Lemma 12.** *The $\mathsf{evc}(G)$ is at least k, where $k \in \mathbb{N}$ such that every vertex of G belongs to a weakly good configuration of size at most k. This means that a graph G has $\mathsf{evc}(G) = k$ only if for each $v \in V(G)$, there exists a weakly good configuration $\mathcal{C}_v$ corresponding to a vertex cover S_v such that $\mathcal{C}_v$ has k guards and $v \in S_v$. In particular, a graph G is Spartan only if for each $v \in V(G)$, there exists a minimum-sized and weakly good vertex cover S_v such that $v \in S_v$.*

Proof. Suppose a graph G has $\mathsf{evc}(G) = k$, and there exists $v \in V(G)$ such that there does not exist a weakly good configuration with k guards with a guard on v. Without loss of generality, we can assume that the defender starts with a configuration with a guard on v, because the attacker can always attack an edge adjacent to v, and ensure that at least one guard comes to v. As there exists no weakly good configuration with k guards, which has a guard on v, the configuration $\mathcal{C}$ formed by the guards cannot be a weakly good configuration. If the guards do not occupy a vertex cover, some edge must be vulnerable and the attacker can attack that edge. So, we can assume that the set S of vertices occupied by the guards in the configuration $\mathcal{C}$ forms a vertex cover. Therefore, there exists a weakly bad subset T of the unoccupied vertices such that if $C_1, C_2, \ldots, C_\ell$ are the connected components of $G[V(G) \setminus T]$, we must have a C_i (where $i \in [1, \ell]$), which contains $\mathsf{mvc}(C_i)$ guards. Now, since the components are not adjacent to each other, there exists an edge from a vertex $w \in V(C_i)$ to a vertex $u \in T$ because the graph G is connected. Since no vertex in T has a guard, the guard on w must come to u. Therefore, some edge in C_i will become vulnerable, as there were only $\mathsf{mvc}(C_i)$ many guards in $V(C_i)$, and one guard has moved out of C_i, while no guard from outside can come to a vertex of C_i in one step. Thus, the attacker wins, and this contradicts $\mathsf{evc}(G) = k$. ◀

▶ **Lemma 13.** *Let S be a minimum-sized vertex cover of a graph G, such that there exists a cut vertex x of G such that $x \notin S$. Then S cannot be a weakly good vertex cover.*

Proof. Let x be a cut vertex of a graph G, and let S be a vertex cover of G which does not contain x. Then we show that $\{x\} \subset V(G) \setminus S$ is weakly bad, which will imply that S is not a weakly good vertex cover. Since x is a cut vertex, there exist connected components $C_1, C_2, \ldots, C_\ell$ where $\ell \geqslant 2$ of $V(G) \setminus \{x\}$. If $|S \cap V(C_i)| > \mathsf{mvc}(C_i)$ for all $i \in [1, \ell]$, then $S' = \{x\} \cup S_1 \cup S_2 \cup \ldots \cup S_\ell$ will be a vertex cover of G of size $|S| - \ell + 1$ (where S_i is a vertex cover of C_i of size $\mathsf{mvc}(C_i)$). This is not possible as $|S'| < |S|$ and S is a minimum-sized vertex cover of G. Thus, there exists some $i \in [1, \ell]$ such that $|S \cap V(C_i)| = \mathsf{mvc}(C_i)$, which implies that $\{x\}$ is a weakly bad set. Hence, S is not a weakly good vertex cover. ◀

Thus, with Lemma 12 and Lemma 13, we get the result in [2] that states that if $\mathsf{evc}(G) = \mathsf{mvc}(G)$, then each vertex must belong to a minimum-sized vertex cover which contains all the cut vertices.

We next define a generalization of a bad set, which is used to define the notion of a strongly good vertex cover. We use this notion to give another necessary condition for a graph G to be Spartan. In order to define this, we define the notion of compatible vertex sets (or vertex covers) in a graph. We also give analogous definitions of compatible configurations and strongly good configurations. This gives us a new lower bound for $\mathsf{evc}(G)$.

▶ **Definition 14.** *Two vertex sets (vertex covers)* S_1 *and* S_2 *of a graph* G *are said to be* compatible*, if there exist* $|S_1 \cap S_2|$*—many vertex-disjoint paths between* $S_1 \setminus S_2$ *and* $S_2 \setminus S_1$*.*

▶ **Definition 15.** *Two configurations* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are said to be* compatible*, if there exist* $|\mathcal{C}_1 \cap \mathcal{C}_2|$*—many vertex disjoint paths (counting multiplicity) between* $\mathcal{C}_1 \setminus \mathcal{C}_2$ *and* $\mathcal{C}_2 \setminus \mathcal{C}_1$*. Here we view each configuration as a multi-set over the set of vertices* $V(G)$*.*

Also, when we say $\mathcal{C}_i \setminus \mathcal{C}_j$, we account for multiplicity of each vertex. That is, if v occurs 5 times in $\mathcal{C}_1$ and 3 times in $\mathcal{C}_2$, v will occur 2 times in $\mathcal{C}_1 \setminus \mathcal{C}_2$. Also, when we say vertex-disjoint paths (counting multiplicity) between $\mathcal{C}_1 \setminus \mathcal{C}_2$ and $\mathcal{C}_2 \setminus \mathcal{C}_1$, we mean that if a vertex v occurs multiple times in $\mathcal{C}_1 \setminus \mathcal{C}_2$, each occurrence of v will be an endpoint of exactly one path from $\mathcal{C}_1 \setminus \mathcal{C}_2$. Similarly, if a vertex v occurs multiple times in $\mathcal{C}_2 \setminus \mathcal{C}_1$, each occurrence of v will be an endpoint of exactly one path to $\mathcal{C}_2 \setminus \mathcal{C}_1$. If we consider the union of all the multisets formed by the intermediate vertices (i.e., the vertices of $\mathcal{C}_1 \cap \mathcal{C}_2$) of each path, the number of times a vertex v occurs in the union, should be less than or equal to the number of times v occurs in $\mathcal{C}_1 \cap \mathcal{C}_2$.

It is clear that two configurations $\mathcal{C}_1$ and $\mathcal{C}_2$ are compatible, if and only if there is a possible movement of guards from $\mathcal{C}_1$ to $\mathcal{C}_2$, where each guard moves at most one step. The guards can rearrange themselves by moving one step from $\mathcal{C}_1 \setminus \mathcal{C}_2$ to $\mathcal{C}_2 \setminus \mathcal{C}_1$ if and only if there exist $|\mathcal{C}_1 \cap \mathcal{C}_2|$—many vertex disjoint paths (counting multiplicity) between $\mathcal{C}_1 \setminus \mathcal{C}_2$ and $\mathcal{C}_2 \setminus \mathcal{C}_1$. For more explanation on the equivalence between the guard movements and vertex-disjoint paths, please refer to Figure 2.

The following is a known result shown in [2].

▶ **Lemma 16** ([2])**.** *Two minimum-sized vertex covers of a graph* G *are always compatible.*

▶ **Definition 17.** *For a vertex cover* S*, a set* $T \subset V(G) \setminus S$ *is* strongly bad*, if there is some connected component* C_i *of* $V(G) \setminus T$ *and some* $v \in N(T) \cap V(C_i)$*, such that no vertex cover of* C_i *of size* $|S \cap V(C_i)| - 1$ *is compatible with* $(S \cap V(C_i)) \setminus \{v\}$*.*

▶ **Definition 18.** *A vertex cover* S *of a graph* G *is said to be* strongly good*, if* $V(G) \setminus S$ *does not have a strongly bad subset.*

Now we analogously define a strongly good configuration.

▶ **Definition 19.** *For a configuration $\mathcal{C}$ of guards on a vertex cover S, a set $T \subset V(G) \setminus S$ is* strongly bad*, if there is some connected component C_i of $V(G) \setminus T$ and some $v \in N(T) \cap V(C_i)$, such that no configuration on a vertex cover of C_i of size $|\mathcal{C} \cap V(C_i)| - 1$ is compatible with $(\mathcal{C} \cap V(C_i)) \setminus \{v\}$.*

▶ **Definition 20.** *A configuration $\mathcal{C}$ on a vertex cover S of a graph G is said to be* strongly good*, if $V(G) \setminus S$ does not have a strongly bad subset.*

Now we make an observation which relates the notions of weakly good and strongly good configurations (vertex covers).

▶ **Lemma 21.** *A strongly good configuration (vertex cover) is a weakly good configuration (vertex cover).*

Proof. We prove the contrapositive of this statement, i.e., a configuration on a vertex cover which is not a weakly good configuration, is also not a strongly good configuration. Let $\mathcal{C}$ be a configuration on a vertex cover S, such that $\mathcal{C}$ is not weakly good. Therefore, there exists a non-empty $T \subseteq I = V(G) \setminus S$, such that if $C_1, C_2, \ldots, C_\ell$ are the connected components of $G[V(G) \setminus T]$, there must exist a C_i (where $i \in [1, \ell]$) such that the number of guards in C_i (in $\mathcal{C}$) is equal to $mvc(C_i)$. Since T is non-empty and the graph G is connected, there exists $v \in V(C_i)$ such that v is adjacent to T and since S is a vertex cover v is in S (and has exactly one guard in $\mathcal{C}$ as the number of guards in C_i (in $\mathcal{C}$) is equal to $mvc(C_i)$). The configuration $\mathcal{C} \cap C_i \setminus \{v\}$ is not compatible with any vertex cover of C_i, as the number of guards in $\mathcal{C} \cap C_i \setminus \{v\}$ is less than $mvc(C_i)$. Thus the set T is a strongly bad set, and $\mathcal{C}$ is not a strongly good configuration.

The same proof works for vertex covers, if we consider a vertex cover as a configuration with one guard per vertex. ◀

▶ **Lemma 22.** *The $evc(G)$ is at least k, where $k \in \mathbb{N}$ such that every vertex of G belongs to a strongly good configuration of size at most k. This means that graph G has $evc(G) = k$, only if for each $v \in V(G)$, there exists a k–sized strongly good configuration $\mathcal{C}_v$ on a vertex cover S_v, such that $v \in S_v$. In particular, a graph G is Spartan if for each $v \in V(G)$, there exists a minimum-sized and strongly good vertex cover S_v, such that $v \in S_v$.*

Proof. Suppose that there exists a graph G with $evc(G) = k$ and a vertex v, such that there exists no k–sized strongly good configuration on a vertex cover containing v. Without loss of generality, we can assume that the initial configuration has at least one guard on v, because if not, the attacker can force a guard to come to v by attacking an edge adjacent to v. If the vertices occupied by the guards in the resulting configuration do not form a vertex cover, then there exists some edge which is vulnerable, and hence the attacker wins. Therefore, the resulting configuration must have guards on a vertex cover. Since there is no k–sized strongly good configuration $\mathcal{C}_v$ on a vertex cover S_v such that $v \in S_v$, for the configuration $\mathcal{C}$ formed by the guards on the vertex cover S, there exists a strongly bad subset $T \subset V(G) \setminus S$. Hence, there is some connected component C_i of $V(G) \setminus T$ and some $u \in N(T) \cap V(C_i)$ such that no configuration on a vertex cover of C_i of size $\mathcal{C} \cap V(C_i) - 1$ is compatible with $S \cap V(C_i) \setminus \{u\}$. Suppose the attacker attacks an edge joining u to a vertex in T, the guard on u is forced to move to T. No guard from a vertex outside C_i can come to C_i and the guards in C_i cannot rearrange themselves to form a vertex cover of C_i. Therefore, some edge will be vulnerable no matter how the guards rearrange themselves, which contradicts $evc(G) = k$.

Therefore, a graph G has $evc(G) = k$, only if for each $v \in V(G)$, there exists a k–sized strongly good configuration $\mathcal{C}_v$ on a vertex cover S_v where $v \in S_v$. ◀

4 König–Egerváry Graphs

A graph G is said to be a *König–Egerváry* graph, if the size of a smallest vertex cover of G is equal to the size of the largest matching of G, i.e., $\mathsf{mvc}(G) = \mathfrak{mm}(G)$. This class is a natural and strict[1] generalization of bipartite graphs. In this section, we derive the necessary and sufficient condition for a König-Egerváry graph to be Spartan. Before that, we have a corollary of Lemma 12, which gives a sense of how a Spartan graph will look like. The proof is given in the Appendix.

▶ **Corollary 23.** *If a graph G with more than one vertex is Spartan and I is an independent set of G which is not maximal, then $|N(I)| > |I|$.*

The main result of this section is the following:

▶ **Theorem 1.** *A König-Egerváry graph G is Spartan if and only if it is bipartite and essentially elementary.*

Proof. As described in Lemma 4, it suffices to look at connected graphs i.e., it will be sufficient to prove that a connected König-Egerváry graph G is Spartan if and only if it is bipartite and elementary. The reverse direction has already been proved in [1]. Thus we need to only show that if a connected König-Egerváry graph G is Spartan, then it is bipartite and elementary.

Since G is Spartan, let S be a minimum-sized vertex cover of G, which is an eternal vertex cover configuration, and let $I = V(G) \setminus S$ be the corresponding independent set. By Lemma 5, there exists a matching M from I to S, which saturates I. This means that $|S| \geqslant |I|$. Since S is a minimum-sized vertex cover, each edge of any matching, and hence a maximum matching of G must have at least one endpoint in S. Suppose some edge of a maximum matching has two endpoints in S, then $|S| > \mathfrak{mm}(G)$, which contradicts the assumption that G is a König-Egerváry graph. Therefore, every edge of a maximum matching of G has exactly one endpoint in S. Thus, $|S| \leqslant |V(G) \setminus S|$ i.e., $|S| \leqslant |I|$. Hence we obtain $|S| = |I| = {}^{V(G)}/_{2}$, and M is a perfect matching of G.

Now, consider the bipartite graph H with $V(H) = V(G)$, and $E(H) = E(G) \setminus E(G[S])$. That is, H has the same vertex set as that of G, and the edge set of H consists of edges of G going across from S to I. Now $V(H) = S \cup I$, such that S and I are independent sets in H, and M is a perfect matching of H. Suppose there exists $T \subsetneq I$, such that $|N(T)| = |T|$ in H. Since I is an independent set in G as well, we have $|N(T)| = |T|$ in G. Since $T \subsetneq I$, T is an independent set of G which is not maximal. Thus, by Corollary 23, we get that G is not Spartan, which is a contradiction. Therefore, H must be elementary, by Lemma 2.

Thus, we know that H can have only the two minimum-sized vertex covers S and I (using an alternate definition of elementary bipartite graph from [5]). Since $E(H) \subseteq E(G)$, any vertex cover of G must be a vertex cover of H. Thus, G cannot have any minimum-sized vertex cover other than S and I. If there is an edge of G with both its endpoints in S, then I is not a vertex cover of G and thus S is the only minimum-sized vertex cover of G. For $v \in I$, there does not exist a minimum-sized vertex cover of G which contains v. Thus, using Lemma 3, we get that G is not Spartan, which is a contradiction. Thus, there cannot be any edge with both the endpoints in S. Thus, $E(G) = E(H)$ and we already know that $V(G) = V(H)$ and H is bipartite and elementary. Therefore, G is bipartite and elementary. ◀

[1] There are König-Egerváry graphs which are non-bipartite as well. For example consider G with $V(G) = \{a_1, b_1, a_2, b_2\}$ and $E(G) = \{a_1a_2, a_1b_1, a_2b_2, a_1b_2, a_2b_1\}$. This graph has a perfect matching $\{a_1b_1, a_2b_2\}$ and thus $\mathfrak{mm}(G) = 2$ and thus $\mathsf{mvc}(G) \geqslant 2$. But $\{a_1, a_2\}$ forms a vertex cover of G of size 2 thus $\mathsf{mvc}(G) = \mathfrak{mm}(G)$ and G is not bipartite because it contains a triangle.

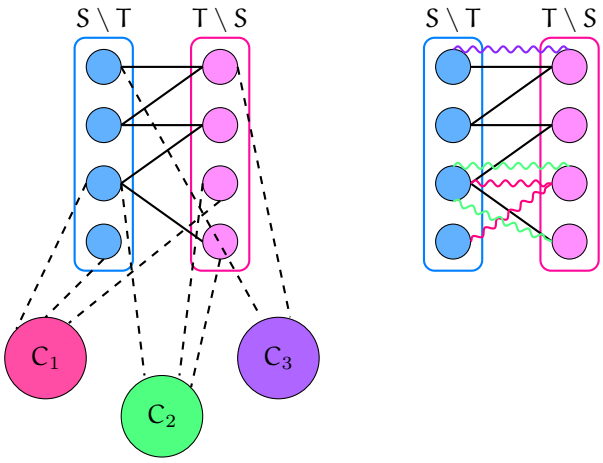


Figure 3 Depicting the construction of $\mathfrak{h}_G(S,T)$ where S and T are two minimum sized vertex covers of G. The connected components of $S \cap T$ are C_1 (red), C_2 (green) and C_3 (purple). The edges is G from $S \setminus T$ to $T \setminus S$ are also there in $\mathfrak{h}_G(S,T)$ as "real edges". For every u in $S \setminus T$ and v in $T \setminus S$ such that u and v are both adjacent to a vertex in C_1, there is a red wavy "helper" edge in $\mathfrak{h}_G(S,T)$, and similarly there are green and purple helper edges corresponding to C_2 and C_3.

5 General Graphs

Before stating the main result of this section, we need the definition of an auxiliary graph, defined for a pair of vertex covers of a graph G. Also, in this section, as we are dealing with Spartan-ness, we are dealing with configurations of guards of size $\mathsf{mvc}(G)$. If there is more than one guard on some vertex of a configuration, the set of vertices occupied by guards does not form a vertex cover, and this configuration can be attacked by the attacker immediately. Hence, we look at only those configurations with one guard per vertex.

▶ **Definition 24.** *Let S and T be two vertex covers of a graph G. Let $H_1, \ldots, H_k$ be the connected components of $G[S \cap T]$. For i, where $1 \leqslant i \leqslant k$, we say that two vertices $u \in S \setminus T$ and $v \in T \setminus S$ are* pseudo-adjacent via i *if both u and v are adjacent to some vertex in $V(H_i)$.*

Let $D = (S \setminus T) \cup (T \setminus S)$. We define two subsets of $D \times D$:

$$E_1 := \{(u,v) \mid u \in S \setminus T, v \in T \setminus S, uv \in E(G)\},$$

and

$$E_2 := \{(u,v) \mid u \in S \setminus T, v \in T \setminus S, uv \text{ are pseudo-adjacent via } i \text{ for some } i\}.$$

We use $\mathfrak{h}_G(S,T)$ to denote the graph $(D, E_1 \uplus E_2)$.

Note that $\mathfrak{h}_G(S,T)$ may have multi-edges. In the context of this graph, we refer to the edges in E_1 as *real* edges and the edges in E_2 as *helper* edges.

Note that the helper edges can be naturally partitioned into k sets $E_2^{(1)}, \ldots, E_2^{(k)}$, where $E_2^{(i)}$ consists of the helper edges that are pseudo-adjacent via i. For convenience, we refer to the edges in $E_2^{(i)}$ as being edges of color i.

The following can be easily observed:

▶ **Lemma 25.** *The graph $\mathfrak{h}_G(S,T)$ is bipartite for a graph G with at least one edge.*

Proof. Since $S \setminus T$ and $T \setminus S$ are subsets of independent sets $V(G) \setminus T$ and $V(G) \setminus S$ respectively, the graph $\mathfrak{h}_G(S,T)$ is bipartite. ◀

▶ **Theorem 26.** *A graph* G *is Spartan if and only if there exists a non-empty family* $\mathcal{F}$ *of minimum-sized vertex covers such that the following condition holds:*

For every $S \in \mathcal{F}$ *and for every edge* $uv \in G$ *for which* $u \in S$ *and* $v \notin S$*, there exists* $T \in \mathcal{F}$ *such that* $v \in T$ *and:*

(1) *either* $u \notin T$ *and there is a perfect matching in* $\mathfrak{h}_G(S,T)$ *which contains the edge* uv*,*

(2) *or* $u \in T$*, and there is a perfect matching in* $\mathfrak{h}_G(S,T)$*, where the matched partner of* v*, say* w*, has a neighbor – in* G *– among the vertices of* X*, where* X *is the connected component of* $G[S \cap T]$ *that contains* u*.*

Proof. First, assume that G is Spartan. Then, there is a strategy for indefinite defense of G with $\mathsf{mvc}(G)$ guards. Let $\mathcal{F}$ be the set of all vertex covers that are used in the strategy: note that they are all of minimum size by definition. Since G is Spartan, $\mathcal{F}$ is indeed non-empty.

Consider a vertex cover $S \in \mathcal{F}$ and an edge uv such that $u \in S$ and $v \notin S$.

As S occurs in the strategy of the defender, there is a way to defend an attack on an edge when guards occupy the vertices of S (one guard on each vertex). We attack the edge uv here and observe the promised defense: let us say that the guards are positioned on the vertex cover T after the defense is executed. Clearly, $T \in \mathcal{F}$, by definition. Depending on whether $u \in T$ or not, we can conclude that either (1) holds or (2) does (respectively), by tracing the movement of the guards from vertices in $S \setminus T$ to vertices in $T \setminus S$.

Suppose $u \notin T$, we show that (1) holds. Let $S' = S \setminus T$, i.e., the set of vertices which had a guard before the attack and do not have a guard after the defense is completed. Let $T' = T \setminus S$, i.e., the set of vertices which had a guard after the defense is completed and do not have a guard before the attack. Let $|S'| = |T'| = k$. Then there must be a collection $\mathcal{P}$ of k vertex disjoint paths from S' to T' (with the starting vertex of each path from S', end vertex in T', and each intermediate vertex from $S \cap T$), and one of these paths must be the edge uv. We show how the collection $\mathcal{P}$ is obtained. Each path is obtained by tracing the movement of each individual guard. The guard on u is forced to move to v by the attacker and hence the edge uv is traced by a guard. Thus $uv \in \mathcal{P}$. Similarly look at the movement of a guard on a vertex say $u_1 \neq u$ of S'. This guard moves to some vertex u_2. If $u_2 \in T$, $u_1u_2 \in \mathcal{P}$. Otherwise $u_2 \in S \cap T$ as the guard can only move for one step after each attack and hence u_2 must have a guard both before and after the attack. The guard which previously was on u_2 moves to some u_3 (as both S and T are vertex covers hence there must be only one guard per vertex after the reconfiguration has been done). Now again if $u_3 \in T$, $u_1u_2u_3 \in \mathcal{P}$. Otherwise $u_3 \in S \cap T$ and the guard which was previously on u_3 must move to some u_4. This process will only stop when a guard moves to a vertex which did not already have a guard, i.e., a vertex in T'. We obtain k paths by tracing the movement of each of the k guards in S' (including the guard moving from u to v). It is clear that a vertex v in S' or T' cannot belong to two paths because this will indicate that two guards started from the vertex v (if $v \in S'$) or two guards ended up at v (if $v \in T'$). A vertex $v \in S \cap T$ cannot belong to two paths in $\mathcal{P}$, because this will indicate that this vertex has at least two guards after the reconfiguration is done which is not possible. Also, each path in $\mathcal{P}$ which contains more than one edge must have all the intermediate vertices from the same connected component of $S \cap T$. This is because the intermediate vertices contain a guard both before and after the reconfiguration and hence lie in $S \cap T$. Each guard can move only to a neighboring vertex, the intermediate vertices of a path in $\mathcal{P}$ also form a path in $S \cap T$. Thus a path $u_1u_2 \dots u_t \in \mathcal{P}$ where $t > 2$ implies $u_1 \in S'$, $u_t \in T'$ and $u_2, u_3, \dots, u_{t-1} \in V(H_i)$, for some i. Therefore, in the graph $\mathfrak{h}_G(S,T)$, there exists a helper edge of color i between u_1 and u_t. Also an edge in $\mathcal{P}$ corresponds to a real edge in $\mathfrak{h}_G(S,T)$. Thus, each path in $\mathcal{P}$ gives an edge in $\mathfrak{h}_G(S,T)$. The edges in $\mathfrak{h}_G(S,T)$ corresponding to the paths in $\mathcal{P}$ form a perfect matching, because each

vertex in S' is adjacent to exactly one edge corresponding to a path in $\mathcal{P}$. This is because there is only one guard on each vertex of S' before the attack and each vertex in S' has no guard after the attack. Thus, $\mathfrak{h}_G(S,T)$ contains a perfect matching containing the edge uv and condition (1) holds.

Now suppose $u \in T$, we show that (2) holds. Let S' and T' be the same as above, and similarly $k = |S'| = |T'|$. By the same reasoning as the previous case, we have k vertex disjoint paths in G, which correspond to a perfect matching M in $\mathfrak{h}_G(S,T)$. Now as $u \in S$ and $u \in T$, a guard is present on u before the attack and after the reconfiguration is complete. Since T is obtained from S by applying the winning strategy for a defense after the attacker attacks the edge uv, the guard on u must move to v while reconfiguring from S to T. Since a guard is present on u after the attack, there must exist a path $u_1u_2\dots u_t$ in $\mathcal{P}$, where $t > 2$, such that $u_1 \in S'$, $u_{t-1} = u$ and $u_t = v$. Also, $u_2, u_3, \dots, u_{t-1}$ belong to the same connected component of H_i. This path corresponds to the following movement: A guard on a vertex u_1 of S' moves to a vertex u_2 of $S \cap T$. The guard on u_2 moves to u_3 of $S \cap T$, and so on, till the guard on u_{t-2} moves to u, and the guard on u moves to v. Clearly, the vertices $u_2, u_3, \dots, u_{t-1}$ must belong to the same connected component of $S \cap T$, because a guard can only move to a neighboring vertex. The edge u_1v is a helper edge in $\mathfrak{h}_G(S,T)$ and lies in M. The vertex u_1 has a neighbor u_2 in the connected component of $G[S \cap T]$ that contains u (because there is a path $u_2u_3\dots u_{t-1} = u$ in $G[S \cap T]$). Hence, (2) holds.

Thus, we have shown that when G is Spartan, the family $\mathcal{F}$ of all vertex covers used in a strategy of the defender; satisfies the condition in Theorem 26.

Now we show the converse, i.e., if a graph G has a family of minimum-sized vertex covers $\mathcal{F}$ which satisfy the condition in Theorem 26, then G is Spartan. For this, we show that if the guards occupy a vertex cover $S \in \mathcal{F}$ and an arbitrary edge uv is attacked, the guards can reconfigure themselves, such that at least one guard moves across the attacked edge uv, and the final positions of the guards form a vertex cover $T \in \mathcal{F}$.

Since S is a vertex cover, there cannot be any edge with both the endpoints outside S. If an edge uv such that $u, v \in S$ is attacked, the guard on u moves to v, and the guard on v moves to u. The attack is defended and the configuration S is restored. Therefore, we need to consider the attack on an edge uv such that $u \in S$ and $v \notin S$.

By the given condition, there exists a vertex cover $T \in \mathcal{F}$ such that and $v \in T$ such that:

(1) either $u \notin T$ and there is a perfect matching in $\mathfrak{h}_G(S,T)$ which contains the edge uv,

(2) or $u \in T$, and there is a perfect matching in $\mathfrak{h}_G(S,T)$, where the matched partner of v, say w, has a neighbor – in G – among the vertices of X, where X is the connected component of $G[S \cap T]$ that contains u.

Suppose (1) holds. We show that it is possible to reconfigure the guards from S to T such that one guard moves across uv. Let $S' = S \setminus T$, $T' = T \setminus S$ and $|S'| = |T'| = k$. It is enough to show that there is a collection $\mathcal{P}$ containing uv of k vertex disjoint paths from S' to T' with all the intermediate vertices in $S \cap T$. A path $u_1u_2\dots u_t$ in G where $u_1 \in S'$, $u_t \in T'$ and $u_2, u_3, \dots, u_{t-1} \in S \cap T$ will represent the movement of a guard from u_1 to u_2, the movement of the guard previously on u_2 to u_3, and so on up to the movement of the guard previously on u_{t-1} to u_t.

If there exists a perfect matching M in $\mathfrak{h}_G(S,T)$ such that it consists of possibly some real edges and at most one helper edge of each color, then there exists a collection $\mathcal{P}$ of k vertex disjoint paths from S' to T' with intermediate vertices in each path from $S \cap T$. We show that a path in $\mathcal{P}$ can be obtained from each edge of M. A real edge in $\mathfrak{h}_G(S,T)$ is also an edge in G. Thus, for each real edge e in M, add e to $\mathcal{P}$. A helper edge uv of color i implies the

existence of a path $u_1(=u)u_2u_3\dots u_t(=v)$ in G where $t>1$ and $u_2,u_3,\dots,u_{t-1}\in V(H_i)$, where H_i is a connected component of $S\cap T$. For each helper edge $e=wz$ in M of color i, add a path $u_1(=w)u_2u_3\dots u_t(=z)$ in G where $t>1$ and $u_2,u_3,\dots,u_{t-1}\in V(H_i)$ to $\mathcal{P}$. Clearly, there are k paths in $\mathcal{P}$ (one path obtained from each edge of M). We show that the paths in $\mathcal{P}$ are vertex disjoint. Since M is a matching, the endpoints of each edge in M are distinct. Hence the endpoints of all the k paths are distinct. Since all the intermediate vertices of each path are distinct and each path of length at least 2 is obtained from a helper edge of different color (as there are no two helper edges of the same color in M), each path obtained from a helper edge in M has intermediate vertices from distinct components of $G[S\cap T]$. Thus all the intermediate vertices of each path in $\mathcal{P}$ are disjoint.

Now we show that there exists a perfect matching in $\mathfrak{h}_G[S,T]$ which contains possibly some real edges and at most one helper edge of each color. By Lemma 16, there exists a perfect matching from S' to T' in G. This means that there exists a perfect matching M_p in $\mathfrak{h}_G(S,T)$ which consists of only real edges. Let M be the perfect matching in $\mathfrak{h}_G(S,T)$ such that M contains uv (exists by condition (1)), and the size of $M\cap M_p$ is as large as possible. Now, consider the graph H in $\mathfrak{h}_G(S,T)$ such that $V(H)=V(\mathfrak{h}_G(S,T))$ and $E(H)=M\cup M_p$. Since M and M_p are both perfect matchings, H is a union of edges and even cycles.

▷ Claim 27. There can be at most one cycle in H.

Proof. If there are two distinct cycles C_1 and C_2 in H, at most one of them can contain the edge uv. Therefore, without loss of generality, we can assume that $uv\notin E(C_1)$. Suppose C_1 is a cycle and $C_1=u_1v_1u_2v_2\dots u_tv_tu_1$, where $M_1:=\{u_1v_1,u_2v_2,\dots,u_tv_t\}\subset M$ and $M_2:=\{v_1u_2,v_2u_3,\dots,v_{t-1}u_t,v_tu_1\}\subset M_p$ with $t>1$. The matching $M':=M\setminus M_1\cup M_2$ has strictly more intersection with M_p than M, and also contains the edge uv, which is a contradiction. ◁

If H has no cycle, then $M=M_p$, which means that all the edges of M are real, and we are done. Now, we consider the case where M has exactly one even cycle C. If $uv\notin E(C)$, then we get a contradiction by the same argument as the claim above. Therefore, let $C=u_1(=u)v_1(=v)u_2v_2\dots u_tv_tu_1$. Here $M_1:=\{u_1v_1,u_2v_2,\dots,u_tv_t\}\subset M$ and $M_2:=\{v_1u_2,v_2u_3,\dots,v_{t-1}u_t,v_tu_1\}\subset M_p$. Also, $\{u_1,u_2,\dots,u_t\}\subset S'$ and $\{v_1,v_2,\dots,v_t\}\subset T'$. This is because $u\in S'$ and $\mathfrak{h}_G(S,T)$ is bipartite (by Lemma 25). All the edges in M which are not in M_1, are also the edges of M_p, and hence are real. Next, we show that M cannot contain two (or more) helper edges of the same color.

▷ Claim 28. M can contain at most one helper edge of each color.

Proof. Suppose M contains two helper edges of the same color (say i). These two edges must lie in M_1, because the edges of M outside M_1 are real. Let these two edges be u_pv_p and u_qv_q, where $1<p<q\leqslant t$. (As uv is a real edge, we have $p\neq 1$.) Therefore, there must exist edge u_pv_q of color i. This is because u_p and v_p are both adjacent to some vertex in $V(H_i)$, and similarly u_q and v_q are both adjacent to some vertex in $V(H_i)$. Hence, u_p and v_q are adjacent to some vertex in $V(H_i)$. Also, $M_3:=\{v_pu_{p+1},v_{p+1}u_{p+2},\dots,v_{q-1}u_q\}\subset M_p$, hence the edges in M_3 are real edges of $\mathfrak{h}_G(S,T)$. Let $M_4=\{u_pv_p,u_{p+1}v_{p+1},\dots,u_qv_q\}$ and $M_5=(M_1\setminus M_4)\cup M_3\cup\{u_pv_q\}$. Consider the perfect matching $M'=(M\setminus M_1)\cup M_5$. It can be checked that M' is a perfect matching which contains uv, and has greater intersection with M_p than M, which is a contradiction. ◁

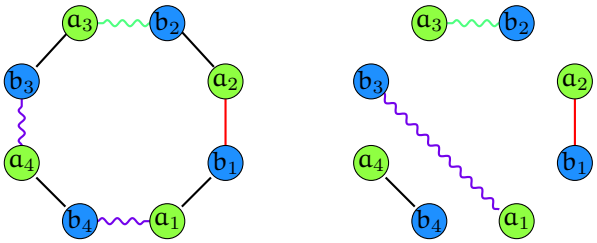


Figure 4 Demonstrating the exchange argument from the proof.

Thus, we have shown the existence of a perfect matching between S' and T' containing the edge uv in $\mathfrak{h}_G(S,T)$, which contains at most one helper edge of each color. This implies that there is a collection of $|S'|$ vertex-disjoint paths from S' to T', such that one path in this collection is the edge uv. Therefore, the guards can reconfigure from S to T (such that one guard moves across uv).

Now, suppose (2) holds. Since w has a neighbor in the same connected component $X = C_i$ (say) of $G[S \cap T]$ that contains u, this means that there exists a path $(w =)u_1u_2 \dots u_{t-1}(= u)u_t(= v)$, where $t > 1$, such that $u_2, u_3, \dots, u_{t-1}(= u)$ belong to $V(C_i)$, that is, wv is a helper edge of color i in $\mathfrak{h}_G(S,T)$. As we have seen above, a matching in $\mathfrak{h}_G(S,T)$, with at most one helper edge of each color, can be used to show the existence of $|S \setminus T|$ vertex-disjoint paths from $S' = S \setminus T$ to $T' = T \setminus S$, with the intermediate vertices from $S \cap T$. This shows that, it is possible to reconfigure the guards from S to T.

Now, if we have a perfect matching in $\mathfrak{h}_G(S,T)$, containing wv with at most one helper edge of each color, it can be seen that it is possible to reconfigure the guards from S to T such that at least one guard moves across uv. This is because, we have already seen that such a matching represents vertex-disjoint paths in G, and particularly, the edge wv represents the path $(w =)u_1u_2 \dots u_{t-1}(= u)u_t(= v)$. This means that, it is possible to move the guard on w to u_2, the guard previously on u_2 to u_3, and so on up to the guard previously on u_{t-2} to u, and the guard previously on u to v. Thus, the existence of a perfect matching in $\mathfrak{h}_G(S,T)$ containing wv with at most one helper edge of each color, is enough to show that the defender can defend an attack on the edge uv, and reconfigure the guards to a vertex cover T, while starting from a vertex cover S. It remains to show the existence of such a perfect matching in $\mathfrak{h}_G(S,T)$.

By Lemma 16, we know that there exists a perfect matching M_p between S' and T' in $\mathfrak{h}_G(S,T)$, which consists of real edges only. Let M be a perfect matching which contains a helper edge of color i adjacent to v (where $u \in V(C_i)$ for the component C_i of $G[S \cap T]$) such that $M \cap M_p$ is as large as possible. It can be seen by a similar argument to the case (1) that the graph H, with $V(H) = S' \cup T'$ and $E(H) = M \cup M_p$, can have at most one cycle (even length).

If H has no cycle, then $M = M_p$, which is not possible because M has only real edges and M_p has at least one helper edge of color i. Let the (exactly one) cycle in H be given by $C = u_1v_1u_2v_2 \dots u_tv_tu_1$, where $M_1 := \{u_1v_1, u_2v_2, \dots, u_tv_t\} \subset M$, and $M_2 := \{v_1u_2, v_2u_3, \dots, v_tu_1\} \subset M_p$. Again, if $wv \notin M_1$, then $M' = M \setminus M_1 \cup M_2$ is a perfect matching containing a helper edge wv (which is of color i and adjacent to v) and $M' \cap M_p$ has greater size than $M \cap M_p$, which is a contradiction. Therefore, $wv \in M_1$. Also, if M_1 contains any two helper edges of the same color (other than wv), then we can use an exchange argument similar to the case (1) shown above to get a contradiction.

It remains to show that C cannot have an edge of color i other than wv. Without loss of generality, let $w = u_1$ and $v = v_1$. Let $u_p v_p$ for some $p > 1$ be the other helper edge of color i. Also, recall that similarly to case (1), $\{u_1, u_2, \ldots, u_t\} \subset S'$ and $\{v_1, v_2, \ldots, v_t\} \subset T'$. This is because $u \in S'$, and $\mathfrak{h}_G(S, T)$ is bipartite (by Lemma 25). This means that $u_p \in S'$, and we already know that $v \in T'$. Since $u_p v_p$ is an edge of color i, u_p is adjacent to $V(C_i)$ in G. We already know that v is adjacent to $V(C_i)$ in G. Therefore, there exists a helper edge $u_p v$ of color i in $\mathfrak{h}_G(S, T)$. Let $M_3 := \{v_2u_1, v_3u_2, \ldots, v_p u_{p-1}, u_p v\}$, all these edges exist in $\mathfrak{h}_G(S, T)$ because the edges in M_3 other than $u_p v$ are from M_p (as seen in the above paragraph) and hence are, in fact, real edges and we have already seen the existence of the edge $u_p v$ which is a helper edge of color i. Let $M_4 := \{u_1v_1, u_2v_2, \ldots, u_p v_p\} \subset M_1$. Let $M_5 := (M_1 \setminus M_4) \cup M_3$. The perfect matching $M' = (M \setminus M_1) \cup M_5$ contains a helper edge $u_p v$ of color i (adjacent to v) and has a greater intersection (in size) with M_p compared to M, which is a contradiction. Therefore, the guards on S can be reconfigured to T while defending the attack on the edge uv. Thus, we have shown that it is always possible for the guards to reconfigure between the vertex covers in $\mathcal{F}$ while defending every attack; thus the graph G is Spartan. ◀

6 Concluding Remarks

In this paper, we give a necessary and sufficient condition for a graph G to be Spartan, i.e., to satisfy $\mathsf{evc}(G) = \mathsf{mvc}(G)$. There are several directions to be pursued further. An important question is whether the complexity of checking whether a given graph G is Spartan is less than that of computing $\mathsf{evc}(G)$. In terms of checking whether $\mathsf{evc}(G) = \mathsf{mvc}(G)$, although we have a complete characterization, the question of finding a simpler characterization remains open. In particular, it would be interesting to know whether there exists a graph G, such that every vertex of G belongs to a strongly (weakly) good vertex cover, but $\mathsf{evc}(G) \neq \mathsf{mvc}(G)$. If not, it would be good to know the proof that this condition is indeed sufficient to guarantee $\mathsf{evc}(G) = \mathsf{mvc}(G)$. Also, we know that there exist weakly good vertex covers that are not strongly good and hence can be destroyed by the attacker (hence, not evc configurations). However, we still do not know whether the condition: "For each vertex v there exists a weakly good minimum vertex cover containing v" implies the following condition: "For each vertex v there exists a strongly good minimum vertex cover containing v".

References

1 Hisashi Araki, Toshihiro Fujito, and Shota Inoue. On the eternal vertex cover numbers of generalized trees. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 98-A(6):1153–1160, 2015. doi:10.1587/TRANSFUN.E98.A.1153.

2 Jasine Babu, L. Sunil Chandran, Mathew C. Francis, Veena Prabhakaran, Deepak Rajendraprasad, and Nandini J Warrier. On graphs whose eternal vertex cover number and vertex cover number coincide. *Discrete Applied Mathematics (in press)*, 2021.

3 Reinhard Diestel. *Graph theory, Sixth Edition.* Springer, 2025.

4 William F. Klostermeyer and Christina M. Mynhardt. Edge protection in graphs. *Australas. J Comb.*, 45:235–250, 2009. URL: http://ajc.maths.uq.edu.au/pdf/45/ajc_v45_p235.pdf.

5 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

6 Neeldhara Misra and Saraswati Girish Nanoti. Spartan bipartite graphs are essentially elementary. In *MFCS*, volume 272 of *LIPIcs*, pages 68:1–68:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.68.

A Appendix

Proof of Claim 6 in Lemma 5. Let T be a minimum-sized vertex cover of G, such that T contains more than $|N(X)|$ vertices from $X \cup N(X)$. Consider the set $T' = \big(T \setminus (X \cup N(X))\big) \cup N(X)$. Clearly, the size of T' is less than the size of T. We show that T' is also a vertex cover of G, which contradicts the fact that T is a minimum-sized vertex cover of G. Any edge with both endpoints outside $X \cup N(X)$ is covered by T, and hence covered by T', because the vertices outside $X \cup N(X)$ which belong to T also belong to T'. Any edge with both endpoints in $X \cup N(X)$ is also covered by T', as $N(X) \subset T'$ and no edge in G can have both endpoints in X, as X is an independent set. An edge with one endpoint in $X \cup N(X)$ and the other endpoint outside $X \cup N(X)$, must have an endpoint in $N(X)$ because a vertex in X cannot be adjacent to a vertex outside $N(X)$ (by the definition of $N(X)$). Thus, such an edge is also covered by T', as $N(X) \subseteq T'$. Thus ,every edge of G has at least one endpoint in T'. This is a contradiction, as G cannot have a vertex cover of size smaller than T. Thus, any minimum-sized vertex cover of G cannot contain more than $N(X)$ vertices from $X \cup N(X)$. ◁

Proof of Corollary 7. Let G be a Spartan graph with more than one vertex, and $|V(G)| = n$. Let I be a maximum independent set of G. Let $S = V(G) \setminus I$. Then, $|N(I)| \geqslant |I|$ by Lemma 5, and thus $|I| \leqslant {}^{|V(G)|}/_{2}$, which means that $|S| \geqslant {}^{|V(G)|}/_{2}$, i.e., $mvc(G) \geqslant {}^{|V(G)|}/_{2}$. ◀

Proof of Corollary 23. Let G be a graph with more than one vertex which is Spartan. Let I be an independent set of G which is not maximal, such that $|N(I)| = |I|$. Denote $N(I)$ by C. Consider an inclusion-wise minimal such set I. The graph H with vertex set $I \cup C$ and edge set $E(G[I \cup C]) \setminus E(C)$ must be bipartite (because I is an independent set) and elementary (using the inclusion-wise minimality of I and Lemma 2). Therefore, using an alternate definition of elementary bipartite graph from [5], I and C are the only two minimum-sized vertex covers of H. Now, any minimum vertex cover S of G cannot contain more than $|I|$ many vertices from $G[I \cup C]$. Suppose not, $S \setminus (I \cup C) \cup C$ is a strictly smaller vertex cover of G (this is a vertex cover, as there are no edges from I to $V(G) \setminus C$, and hence $S \setminus C$).

Now, since G is Spartan, by Lemma 12, every vertex in I belongs to a minimum-sized weakly good vertex cover. Consider a minimum-sized weakly good vertex cover T, which contains some $v \in I$. Clearly, T can contain only $|I|$ vertices from $C \cup I$. Also as T is also a vertex cover of H of size $|I|$ and H has only two minimum vertex covers, $I \subset T$ and $C \cap T = \emptyset$. The graph $G[V(G) \setminus (I \cup C)])$ is non-empty as I is not maximal. If $T \cap (V(G) \setminus (I \cup C))$ is more than $mvc(G[V(G) \setminus (I \cup C)])$, then a smaller vertex cover of G than T can be obtained from the union of C and a minimum vertex cover of $(V(G) \setminus (I \cup C))$. Thus $T \cap (V(G) \setminus (I \cup C))$ is equal to the $mvc(G[V(G) \setminus (I \cup C)])$, which makes C a weakly bad subset for T which is a contradiction. ◀

Finding Small Dijoins in Transitive Closure Time

Chaitanya Nalam
University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak
University of Michigan, Ann Arbor, MI, USA

Abstract

We present a faster algorithm for finding a *minimum dijoin*, a smallest set of edges whose contraction makes a directed graph strongly connected. This problem has been studied since the 1960s [Seshu and Reed 1961] and is dual to finding a maximum sized family of disjoint *dicuts* [Lucchesi and Younger 1978].

Given a directed graph G with n vertices and m edges whose minimum dijoin has size d, our algorithm outputs both a minimum dijoin and a maximum sized family of disjoint dicuts in $O(\mathrm{TC} \cdot d)$ time, where $\mathrm{TC} = \min(mn, n^\omega)$ is the time to compute the transitive closure. This improves upon the state of the art of [Gabow 1993], which requires $O(\mathrm{TC} \cdot \min(m^{1/2}, n^{2/3}))$ time when $d = o(\min(m^{1/2}, n^{2/3}))$. Our result extends to finding a minimum *weighted* dijoin. We achieve this by observing that Frank's algorithm [Frank 1981] can be sped up when warm-started with a 2-approximation solution, which we observed can be computed in near-linear time.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases Graph algorithms, Dijoin, Submodular flow

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.46

Funding *Thatchaphol Saranurak*: Supported by NSF Grant CCF-2238138.

1 Introduction

The primary goal of this work is to give an algorithm with an improved running time for the minimum (weighted) dijoin problem. We define the problem formally.

1.1 Problem definition

In a directed graph $G = (V, E)$ with n vertices and m edges, a *dijoin* is a subset of edges $B \subset E$ whose contraction makes the graph strongly connected. The *minimum dijoin* problem seeks a dijoin of minimum possible size.

The dual of the minimum dijoin problem is to find a *maximum disjoint dicut family*, defined as follows. For any vertex sets $S, T \subseteq V$, let $E(S, T)$ denote the set of edges $(u, v) \in E$ where $u \in S$ and $v \in T$. A *directed cut*, or simply *dicut*, is a subset of edges $D \subset E$ such that there exists a set $S \subset V$ for which $E(S, V \setminus S) = \emptyset$ and $E(V \setminus S, S) = D$. In the maximum disjoint dicut family problem, we want to find a maximum collection of dicuts with pairwise disjoint edge sets.

The weighted variants of these problems involve an edge weight function $w : E \mapsto \mathbb{Z}^+$. The minimum weighted dijoin requires finding a dijoin that minimizes the total weight of its edges. Its dual deals with finding a maximum-sized *w-independent dicut family*, a family of dicuts such that each edge e appears in at most $w(e)$ dicuts within the family.

1.2 Previous work and technical challenges

Enhancing connectivity in directed graphs under minimal edge additions/contractions is a fundamental theme in graph theory. A central problem in this domain is the minimum dijoin problem, first posed by John Runyon (as cited in [23]). Younger [25] and Robertson


© Chaitanya Nalam and Thatchaphol Saranurak;
licensed under Creative Commons License CC-BY 4.0

(unpublished) independently conjectured that the minimum size of a dijoin equals the maximum size of a disjoint dicut family. This fascinating min-max statement was first proved for bipartite graphs by [21] and then proved for general graphs by Lucchesi and Younger in their seminal work [20]. This theorem is a canonical example of problems captured by the submodular flow framework [2], a vast generalization of both minimum-cost flow and matroid intersection. It has played a central role in the development of submodular flow algorithms [26, 22, 5, 8, 9, 10, 14, 3, 7, 18].

Below, we discuss the algorithmic aspect of these problems. All prior results extend naturally to weighted variants. Lucchesi and Younger [19], Karzanov [17], and Frank [4] independently gave the first polynomial-time algorithms for the minimum dijoin problem. Among them, the algorithm by Frank [4] runs in $O(mn^3)$ time. Frank's algorithm maintains a feasible dijoin and a disjoint dicut family and iteratively improves both until their sizes become equal, at which point both are optimal by the min-max theorem. He provided an $O(mn^2)$-time subroutine for improving the solution pair, and showed that the number of iterations is at most the size of the initial feasible dijoin. This quantity is bounded by $n-1$, as edges of any spanning tree form a feasible dijoin, leading to the $O(mn^3)$ time claimed above.

Let $\mathrm{TC} = O(\min(mn, n^\omega))$ denote the time to compute the transitive closure of a directed graph with n vertices and m edges, where ω is the matrix multiplication exponent. Gabow [13, 12] improved the running time of the iterative sub-routine to TC. He [14] further reduced the number of iterations to $O(\min(m^{1/2}, n^{2/3}))$ leading to the state-of-the-art running time of $O(\mathrm{TC} \cdot \min(m^{1/2}, n^{2/3}))$. The algorithm by Gabow [12] for the iterative sub-routine is applicable to a class of more general problems. Subsequent work by [24, 16] gave simpler $O(mn)$-time algorithms for the iterative sub-routine specifically tailored to the dijoin problem.

All of these approaches rely on computing *minimal tight sets* to improve the pair of solutions. Shepherd and Vetta [24] gave a reduction showing that the minimal tight set computation is as hard as computing the transitive closure. Hence, TC effectively becomes the best time bound one can hope for within this framework. This leads us to the following question.

Can we compute a minimum dijoin and a maximum disjoint dicut family in $O(\mathrm{TC})$ time?

Although the dijoin problem is a special case of submodular flow, recent developments in submodular flow algorithms have not yielded improved bounds. In particular, the algorithm by Lee, Sidford and Wang [18] applies only when the input submodular function is defined over a complete set family $\mathcal{F} = 2^V$, whereas the dijoin problem requires a function defined on a more general crossing family where, in particular, $\mathcal{F} \subsetneq 2^V$.

1.3 Our Contributions

Improved algorithm for unweighted dijoin. We give an algorithm with running time $O(\mathrm{TC} \cdot d)$ for solving the minimum dijoin problem, where d is the size of the minimum dijoin and maximum disjoint dicut family. Formally,

▶ **Theorem 1.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$, computes a minimum dijoin and a maximum disjoint dicut family in $O(d \cdot \min(mn, n^\omega))$ time.*

Within the algorithmic framework that relies on computing minimal tight sets, our algorithm achieves the optimal $O(\mathrm{TC})$ time when $d = O(1)$. When $d = o(\min(m^{1/2}, n^{2/3}))$, our algorithm outperforms the fastest algorithm by Gabow [14].

We also extend this result to the weighted case, giving an algorithm to find a minimum weighted dijoin. Formally,

▶ **Corollary 2.** *There exists an algorithm that, given a weighted directed graph $G = (V, E, w)$ where $w : E \mapsto \mathbb{Z}^+$, computes a minimum weighted dijoin in $O(d_{uw} \cdot \min(mn, n^\omega))$ time, where d_{uw} represents the minimum cardinality of all feasible dijoins.*

1.4 Organization

Section 2 presents the two key ingredients needed to obtain an algorithm for the minimum weighted dijoin problem, and then Section 3 combines them, proving Theorem 1 and Corollary 2. We conclude this section by introducing the notation used throughout the rest of the paper.

1.5 Preliminaries

For a directed edge (u, v), we refer to u as the *tail* and v as the *head*. For any two subsets of vertices $S, T \subset V$, let $E(S, T)$ denote the set of directed edges whose head is in T and tail is in S. We write $\delta^-_G(S)$ to denote the set of incoming edges to S, i.e., $E(V \setminus S, S)$, and $\delta^+_G(S)$ for the outgoing edges from S, i.e., $E(S, V \setminus S)$ in the graph G. We omit the subscripts if it is clear from the context. We refer to a set $S \subset V$ with $\delta^+(S) = \emptyset$ and $\delta^-(S) \neq \emptyset$ as the *kernel* of the dicut $\delta^-(S)$.

We begin with some simple structural observations about the dijoin problem. Without loss of generality, we may assume the input graph is weakly connected, i.e., the underlying undirected graph obtained by ignoring edge directions is connected, since disconnected graphs have no feasible dijoin.

We may also assume that G is acyclic. If G contains a directed cycle C, then any vertex reachable to/from a vertex in C can reach or be reached from all vertices in C. Therefore, contracting C preserves reachability and does not affect the minimum dijoin. Repeating this process, we reduce G to a DAG without changing the dijoin size.

This also ensures that G does not contain both an edge and its reverse, as they form a directed cycle of length two.

A directed graph is strongly connected if and only if it contains no dicut. Since contracting an edge in a dicut removes it, a dijoin can be equivalently defined as a subset of edges intersecting every dicut in the graph.

We use G^R to denote the graph obtained by reversing all arcs of G. For a weighted graph $G = (V, E, w)$, $G^R = (V, E^R, w^R)$ where E^R is the set of reverse arcs of E and $w^R : E^R \mapsto \mathbb{Z}^+$ is defined by $w^R(a) = w(a^R)$. The following proposition formalizes the symmetry of the dijoin problem under graph reversal.

▶ **Proposition 3.** *Let $G = (V, E, w)$ be a weighted directed graph, and let $G^R = (V, E^R, w^R)$ be its reverse. Then a subset $B \subseteq E$ is a feasible dijoin in G if and only if $B^R \subseteq E^R$ is a feasible dijoin in G^R, and the total weights of the two solutions are equal.*

Proof. Since contraction and reversal commute, we have $(G^R/B^R)^R = G/B$ for any edge set $B \subseteq E$. Moreover, strong connectivity is preserved under reversal: G is strongly connected if and only if G^R is. Thus, B is a dijoin in G if and only if B^R is a dijoin in G^R. The weight equality follows from the definition of w^R. ◀

We formally define the min-cost s-arborescence problem for future reference.

▶ **Definition 4** (s-arborescence). *Given a graph $G = (V, E, w)$ and a vertex $s \in V$, an* s-arborescence *is a spanning tree of G rooted at s, with no incoming edges into s, where every vertex $v \neq s$ has exactly one incoming edge. The cost of the arborescence is defined as the total weight of all edges of the arborescence.*

▶ **Problem 5** (Min-cost s-arborescence). *Given a directed graph $G = (V, E, w)$ and $s \in V$, find an s-arborescence of minimum cost, if one exists.*

2 Technical overview

In this section, we state the two key ingredients needed for proving Theorem 1: Frank's primal-dual algorithm from [4, 12] for computing the optimal weighted dijoin, and a 2-approximation algorithm for the same problem from [6, 1].

First ingredient

We begin by recalling Frank's primal-dual framework for the minimum weighted dijoin problem, where the primal corresponds to the dijoin problem and the dual to the w-independent dicut family. The algorithm maintains a potential function $p : V \mapsto \mathbb{Z}^+$ as a proxy for the dual solution. (The construction of the corresponding dicut family is deferred.) Frank gives sufficient conditions on a dijoin-potential pair (B, p) that ensure the optimality of both solutions:

Optimality criteria:

$$\forall e = (u, v) \in B, \qquad p(u) - p(v) = w(uv) \tag{1}$$

$$\forall e = (u, v) \in E \setminus B, \qquad p(u) - p(v) \leq w(uv) \tag{2}$$

$$\forall u \in T_B(v), \qquad p(u) \leq p(v) \tag{3}$$

Here, $T_B(v)$ denotes the intersection of all kernels containing v whose corresponding dicuts intersect B in exactly one edge. These are referred to as *minimal tight sets* in Frank's work.

Let B_0 be any spanning tree and p_0 the zero function. Frank starts with (B_0, p_0) as the initial primal-dual pair. Note that (B_0, p_0) already satisfies Equations (2) and (3) and violates only Equation (1). He then describes a subroutine that updates the given (B, p) to a new pair (B', p') that still satisfies Equations (2) and (3) and a strictly smaller number of edges violate Equation (1). We define it below. Let violate(B, p) denote the set of edges that violate Equation (1) with respect to the solution pair (B, p).

▶ **Lemma 6** (Algorithm 4.1 from [4]). *There exists a subroutine, ImprovePair, which takes a directed graph $G = (V, E, w)$, a primal-dual pair (B, p) satisfying Equations (2) and (3), along with an edge $e \in B$ that currently violates Equation (1). It returns a new pair (B', p') such that:*

- *Equations (2) and (3) still hold for the pair (B', p'),*
- violate$(B', p') \subseteq$ violate$(B, p) \setminus \{e\}$

The subroutine runs in $O(n^2 + f(n))$ time, where $f(n)$ denotes the time required to compute $T_B(v)$ for all $v \in V$.

The proof of correctness for this lemma is given in Section 4 of [4], and the running-time analysis in Section 5. To compute $T_B(v)$ for all vertices in Lemma 6, Gabow [12] showed that this can be done in $O(\text{TC})$ time.

▶ **Lemma 7** (Theorem 12.1 from [12]). *Given $G = (V, E)$ and a dijoin B, the sets $T_B(v)$ for all $v \in V$ can be computed in $O(\min(mn, n^\omega))$ time.*

As discussed in the introduction, transitive closure can be reduced to computing minimal tight sets. Hence, this lemma achieves the best possible running time for this task.

Second ingredient

As noted in Lemma 6, the number of initially violated edges determines the number of ImprovePair iterations and hence the overall running time.

While [4, 14] use a spanning tree as the trivial starting solution, we observe that a better dijoin can be obtained in linear time whose size is at most twice that of the optimum. This yields Theorem 1 and Corollary 2.

Bang-Jensen et al. [1] give a 2-approximation algorithm for a generalization of the minimum dijoin problem. We note that their algorithm for the dijoin problem can be implemented in near-linear time and outline the details below.

For this purpose, we define the *rooted* variant of the dijoin problem as follows.

▶ **Problem 8** (Minimum s-Dijoin). *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, an s-dijoin $B \subseteq E$ is a set of edges such that, after contracting them, s can reach all vertices. The minimum s-dijoin problem is to find an s-dijoin with minimum total weight.*

Frank [6] observes that a minimum s-dijoin can be computed in near-linear time by reducing to the min-cost arborescence problem. Note that adding a reverse edge is equivalent to contracting an edge in terms of preserving reachability. Hence an s-dijoin can be equivalently seen as the set of reverse edges which need to be added so that every vertex is reachable from s. Given these observations, the reduction is simple and proceeds as follows.

Given a graph $G = (V, E, w)$, construct the graph $G' = G^0 \cup G^R$, where G^0 is the same graph as G with edge weights set to zero; refer Section 1.5 for the definition of G^R. Compute a min-weight s-out-arborescence T. Observe that the set of reverse edges in T corresponds to an s-dijoin (with respect to the equivalent definition of adding reverse edges) in G and is the minimum-weight set with that property. As the minimum-weight s-arborescence can be found in $O(m + n \log n)$, we have the following theorem.

▶ **Theorem 9.** *There exists an algorithm that takes a weighted directed graph $G = (V, E, w)$ and a vertex $s \in V$ as input and finds a minimum s-dijoin in $O(m + n \log n)$ time.*

It is worthwhile to note that even for the s-dijoin problem, computing minimal tight sets is known to require at least transitive-closure time [24], presenting a fundamental bottleneck for any approach that relies on such computations. Theorem 9 circumvents this barrier by avoiding the computation of minimal tight sets entirely. An intriguing open question is whether a similar approach can be developed for the general minimum dijoin problem, potentially breaking the $O(\mathrm{TC})$-time barrier.

We observe that the dual of the s-dijoin – similarly defined as a maximum w-independent s-dicut family – can also be solved in $O(m + n \log n)$ time. This is discussed in detail in Appendix A, as it is not directly relevant to the current work.

Given Theorem 9, the key insight in [1] is that combining a minimum s-dijoin in G with the reverse of a minimum s-dijoin in G^R results in a 2-approximate dijoin of G. The algorithm is shown below. Let sDijoin denote the algorithm mentioned in Theorem 9.

Algorithm 1 APPROXDIJOIN.

Input : Graph $G = (V, E, w)$
Output : 2-approximate dijoin
Construct $G^R = (V, E^R, w^R)$;
Let s be any vertex in V;
$B_1 = \text{SDIJOIN}(G, s)$;
$B_2 = \text{SDIJOIN}(G^R, s)$;
return $B_1 \cup B_2^R$;

▶ **Theorem 10** ([1]). *There exists an algorithm that, given a weighted directed graph $G = (V, E, w)$, outputs a 2-approximate dijoin in $O(m + n \log n)$ time.*

We now state an immediate corollary of Theorem 10, needed in the next section.

▶ **Corollary 11.** *For any unweighted directed graph, a feasible dijoin of size at most $2d$ (where d is the size of an optimal dijoin) can be found in $O(m + n \log n)$ time.*

3 $O(mnd)$-Algorithm for Minimum Dijoin

This section combines the two ingredients from the technical overview to derive our algorithm and prove Theorem 1 and Corollary 2. Lemma 12 shows how IMPROVEPAIR yields a primal-dual pair (B^*, p^*) satisfying the optimality conditions, and Lemma 13 shows how to construct the corresponding optimal disjoint dicut family. Together, these results establish Theorem 1, which is followed by the proof of Corollary 2.

We will now prove Lemma 12.

▶ **Lemma 12.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$, computes a primal-dual pair (B^*, p^*) satisfying Equations (1)–(3) in $O(d \min(mn, n^\omega))$ time.*

Proof. Let B_0 be a dijoin of size at most $2d$ computed using Corollary 11, and let $p_0 \equiv 0$ be the initial potential function. The pair (B_0, p_0) satisfies Equations (2) and (3), but may violate Equation (1) for up to $|B_0|$ many edges.

Each invocation of IMPROVEPAIR fixes at least one violation without introducing new ones, so at most $|B_0| \leq 2d$ invocations are needed to output a pair, (B^*, p^*), that satisfies the optimality criteria. By Lemma 6 and Lemma 7, each invocation is executed in $O(n^2 + \min(mn, n^\omega))$ time. Thus, the total time is

$$O(m + n \log n) + O(n^2 + \min(mn, n^\omega)) \cdot 2d = O(d \min(mn, n^\omega)).$$ ◀

We now describe how to construct a disjoint dicut family from the optimal solution pair (B^*, p^*) in the following lemma.

▶ **Lemma 13.** *There exists an algorithm that, given an unweighted directed graph $G = (V, E)$ and an optimal solution pair (B^*, p^*), computes a maximum disjoint dicut family in $O(n^2 d + \min(mn, n^\omega))$ time, where d is the number of dicuts in the optimal family.*

This lemma provides the second part of our promised output: a maximum disjoint dicut family. We present a modification to the algorithm by Frank (specifically, the version described at the end of Section 5 in [4]), altering the representation of the dicut family it outputs. This modification is necessary because Frank's original algorithm requires $O(n^4)$ time to construct the optimal dicut family from an optimal potential function, whereas we aim to compute the optimal dual solution within $O(d \cdot \min(mn, n^\omega))$ time.

If D denotes the edge set of a dicut in the family, we output the complement of the corresponding kernel. For any graph G, let $CC(G)$ denote the weakly connected component partition of the vertex set of graph G. The algorithm is as follows:

Algorithm 2 DisjointDicutFamily from [4].

Input : Graph $G = (V, E)$, optimal solution pair (B, p)
Output : Maximum disjoint dicut family

1 Compute $T_B(v)$ for all $v \in V$;
2 Let $p_1, \cdots, p_l$ be the distinct potential values in ascending order;
3 For $i \in [l]$, let $P_i = \{u \mid p(u) = p_i\}$;
4 Initialize $\mathcal{F} = \emptyset$;
5 **for** $i \leftarrow 1$ **to** $l - 1$ **do**
6 Define $V_i = \bigcup_{j=1}^{i} P_j$;
7 Define $G_i = (V_i, E_i)$ where $E_i = \{(x, y) \mid x, y \in V_i, y \in T_B(x)\}$;
8 Let $\mathcal{C} = CC(G_i)$;
9 **for** $C \in \mathcal{C}$ **do**
10 $\mathcal{F} = \mathcal{F} \cup CC(G[V \setminus C])$ where $G[S]$ denotes the induced subgraph on $S \subseteq V$;
11 **return** $\mathcal{F}$;

We will now prove Lemma 13.

Proof of Lemma 13. Given the unweighted directed graph $G = (V, E)$ and the pair of optimal primal dual solutions (B^*, p^*) as inputs to Algorithm 2. It outputs a disjoint dicut family in which each edge occurs in at most one dicut which follows from the optimality condition Equation (1) as explained in [4]. The optimality of this family follows as the size of the family is equal to the cost of the optimal dijoin, as proved in [4].

We now analyze the running time of the algorithm when the graph is unweighted. It takes $O(\min(mn, n^\omega))$ time to compute $T_B(v)$ for all $v \in V$ in line 1 using Lemma 7. Lines 2 and 3 take $O(n \log n)$ to sort and compute the partitions P_i. Defining G_i in line 7 and computing $CC(G_i)$ take at most $O(n^2)$ as there can be $O(n^2)$ edges in the worst case. Note that each iteration of the outer loop adds at least one dicut to $\mathcal{F}$. As the size of the optimal disjoint dicut family is d, $l \leq d + 1$. Hence, it takes $O(n^2 d)$ time overall for line 7 over the course of the algorithm.

For any set $S \subset V$, $G[V \setminus S]$ contains at most m edges. Thus, it takes $O(m)$ time to compute $CC(G[V \setminus C])$ for each component C of G_i. Since each induced subgraph has at least one component, it adds at least one dicut to $\mathcal{F}$. As there can be at most d such computations in total, it takes $O(md)$ time in total during the algorithm for line 10.

Thus, the overall running time is $O(\min(mn, n^\omega) + n \log n + n^2 d + md) = O(n^2 d + \min(mn, n^\omega))$. ◀

We now present the proof of the main result.

Proof of Theorem 1. Given an unweighted directed graph $G = (V, E)$, we first compute the pair of optimal primal-dual solutions (B^*, p^*) using the algorithm from Lemma 12, where B^* is the optimal dijoin. Then, we apply the algorithm from Lemma 13 with inputs G and (B^*, p^*) to obtain the optimal disjoint dicut family.

Computing the optimal primal-dual pair takes $O(d \cdot \min(mn, n^\omega))$ time, and constructing the dicut family takes $O(n^2 d + \min(mn, n^\omega))$ time. The total time remains $O(d \cdot \min(mn, n^\omega))$. ◀

We will now prove Corollary 2 using Theorem 1.

Proof of Corollary 2. Given a weighted graph $G = (V, E, w)$ with $w : E \mapsto \mathbb{Z}^+$, consider the underlying unweighted graph G_{uw} and let d, d_{uw} be the cardinalities of the optimal weighted dijoin and the optimal unweighted dijoin, respectively. We know that $d_{uw} \leq d$ as any weighted dijoin ignoring weights is a feasible dijoin with the same cardinality.

Let B_{uw} be the feasible dijoin obtained by giving G_{uw} as input to Corollary 11 and it guarantees that $|B_{uw}| \leq 2d_{uw}$.

Hence, starting with the solution pair (B_{uw}, p) where $p \equiv 0$ as the potential function, at most $2d$ edges violate Equation (1). From the guarantees of Lemma 6, together with Lemma 7, we know that each iteration takes $O(\min(mn, n^\omega))$ time and reduces one violated edge, so the final pair (B^*, p^*) after at most $|B_{uw}| \leq 2d_{uw}$ iterations, satisfies the optimality criteria which implies optimality; in particular, B^* is an optimal weighted dijoin. The total time taken is at most $O(d_{uw} \cdot \min(mn, n^\omega))$. ◀

References

1 Jørgen Bang-Jensen, Florian Hörsch, and Matthias Kriesell. Complexity of (arc)-connectivity problems involving arc-reversals or deorientations. *Theor. Comput. Sci.*, 973:114097, 2023. doi:10.1016/j.tcs.2023.114097.

2 Jack Edmonds and Rick Giles. A min-max relation for submodular functions on graphs. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 185–204. Elsevier, 1977. doi:10.1016/S0167-5060(08)70734-9.

3 Lisa Fleischer and Satoru Iwata. Improved algorithms for submodular function minimization and submodular flow. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 107–116. ACM, 2000. doi:10.1145/335305.335318.

4 András Frank. How to make a digraph strongly connected. *Comb.*, 1(2):145–153, 1981. doi:10.1007/BF02579270.

5 András Frank. Finding feasible vectors of edmonds-giles polyhedra. *Journal of Combinatorial Theory, Series B*, 36(3):221–239, 1984. doi:10.1016/0095-8956(84)90029-7.

6 András Frank. Connections in combinatorial optimization. *Discrete Appl. Math.*, page 1875, 2012. doi:10.1016/j.dam.2011.09.003.

7 András Frank and Zoltán Miklós. Simple push-relabel algorithms for matroids and submodular flows. *Japan Journal of Industrial and Applied Mathematics*, 29(3):419–439, 2012. doi:10.1007/s13160-012-0076-y.

8 András Frank and Éva Tardos. Generalized polymatroids and submodular flows. *Mathematical Programming*, 42(1):489–563, 1988. doi:10.1007/BF01589418.

9 András Frank and Éva Tardos. An application of submodular flows. *Linear Algebra and its Applications*, 114-115:329–348, 1989. doi:10.1016/0024-3795(89)90469-2.

10 Satoru Fujishige. Submodular functions and optimization. In *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*, page 1. Elsevier, 1990.

11 D. R. Fulkerson. Packing rooted directed cuts in a weighted directed graph. *Math. Program.*, 6(1), 1974. doi:10.1007/BF01580218.

12 H. N. Gabow. Centroids, representations, and submodular flows. *Journal of Algorithms*, 18(3):586–628, 1995. doi:10.1006/jagm.1995.1022.

13 Harold N. Gabow. A representation for crossing set families with applications to submodular flow problems. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 202–211. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313753.

14 H.N. Gabow. A framework for cost-scaling algorithms for submodular flow problems. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 449–458, 1993. doi:10.1109/SFCS.1993.366842.

15 Anupam Gupta. Lecture 2: Arborescences, 2020. Lecture Notes, CMU Course 15-850: Advanced Algorithms.

16 Satoru Iwata and Yusuke Kobayashi. An algorithm for minimum cost arc-connectivity orientations. *Algorithmica*, 56(4):437–447, 2010. doi:10.1007/s00453-008-9179-x.

17 A. V. Karzanov. On the minimal number of arcs of a digraph meeting all its directed cutsets. *Graph Theory Newsletter*, 8(4), 1979. Abstract.

18 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. *CoRR*, abs/1508.04874, 2015. arXiv:1508.04874.

19 C. L. Lucchesi. *A Minimax Equality for Directed Graphs*. Ph.d. thesis, University of Waterloo, 1976.

20 C. L. Lucchesi and D. H. Younger. A minimax theorem for directed graphs. *Journal of the London Mathematical Society*, s2-17(3):369–374, 1978. doi:10.1112/jlms/s2-17.3.369.

21 I. P. McWhirter and D. H. Younger. Strong covering of a bipartite graph. *Journal of the London Mathematical Society*, s2-3(1):86–90, 1971. doi:10.1112/jlms/s2-3.1.86.

22 A. Schrijver. *Min-Max Results in Combinatorial Optimization*, pages 439–500. Springer Berlin Heidelberg, 1983. doi:10.1007/978-3-642-68874-4_18.

23 S. Seshu and M.B. Reed. *Linear Graphs and Electrical Networks*. Addison-Wesley series in behavioral science: Quantitative methods. Addison-Wesley Publishing Company, 1961. URL: https://books.google.com/books?id=mMk-AAAAIAAJ.

24 F.B. Shepherd and A. Vetta. Visualizing, finding and packing dijoins. In David Avis, Alain Hertz, and Odile Marcotte, editors, *Graph Theory and Combinatorial Optimization*, pages 219–254. Springer-Verlag, 2005. doi:10.1007/0-387-25592-3_8.

25 D. H. Younger. *Feedback in a Directed Graph*. Ph.d. thesis, Columbia University, 1963.

26 U. Zimmermann. Minimization on submodular flows. *Discrete Applied Mathematics*, 1982. doi:10.1016/0166-218X(82)90053-1.

A Optimal dual of s-dijoin

In this section, we show how to solve the dual problem of s-dijoin. We begin with some definitions.

▶ **Definition 14** (s-dicut). *A dicut D of the graph G is an* s-dicut *for some vertex $s \in V$ if the kernel corresponding to the dicut contains s.*

The dual problem follows.

▶ **Problem 15** (Maximum w-independent s-dicut family). *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, find a family of s-dicuts with maximum possible size such that, each edge $e \in E$ appears in at most $w(e)$ s-dicuts within the family.*

We give an algorithm that solves this problem in $O(m + n \log n)$ time.

▶ **Theorem 16** (Informal). *There exists an algorithm that computes optimal w-independent s-dicut family in $O(m + n \log n)$ time.*

There is a slight catch in the informal theorem above. The total size of the optimal s-dicut family can be as large as $\Omega(n^2)$ and since we only have $O(m + n \log n)$ time, we have to be focusing on how we represent the output.

To represent a dicut, we can either list its edges or describe the vertex set defining the cut. In this case, we use the vertex-set representation. One can show that the complement of the kernels corresponding to the optimal family of s-dicuts forms a *laminar family.* A family of vertex sets is laminar if any two sets in the family are either disjoint or one contains the other. So, we can represent them using a tree and we output this tree representation of an optimal w-independent s-dicut family in $O(m + n \log n)$ time.

To achieve this we follow a similar strategy as before reducing Problem 15 to the dual of the min-cost s-arborescence problem. The key technical claim is that the algorithm for the dual of the min-cost s-arborescence problem, when run on the graph $G^0 \cup G^R$, produces a w-independent s-dicut family.

We begin by introducing the necessary terminology and then present the dual of the minimum cost s-arborescence.

A *cut* in a directed graph is defined as the set of edges C for which there exists a set $S \subset V$ such that $\delta^+(S) = C$. An *s-cut* is a cut where the corresponding vertex set S contains s. Note that the definitions of s-cut and s-dicut do not correlate in the directionality.

The dual of the min-cost s-arborescence is stated as follows:

▶ **Problem 17** (Maximum w-independent s-cut family)**.** *Given a directed graph $G = (V, E, w)$ and a vertex $s \in V$, find a family of s-cuts of maximum possible size such that each edge is contained in at most $w(e)$ many s-cuts in the family.*

The minimax result relating Problem 5 and Problem 17 is due to Fulkerson, who also provided the first algorithm for constructing an optimal dual solution to the minimum-cost arborescence problem.

▶ **Theorem 18** (Fulkerson branching theorem [11])**.** *The value of the minimum-cost s-arborescence equals the maximum possible size of a w-independent s-cut family.*

We refer to the lecture notes by Gupta [15], who presents an exposition showing how the algorithm for the minimum-cost arborescence implicitly solves the dual problem. He also provides a procedure to recover the dual solution by constructing the corresponding tree representation, thereby effectively proving the following theorem.

▶ **Theorem 19.** *There exists an algorithm that, given a directed graph $G = (V, E, w)$ and $s \in V$, computes a tree representation, $T_{\mathcal{F}}$, of the maximum-sized w-independent s-dicut family $\mathcal{F}$ in $O(m + n \log n)$ time.*

We use Theorem 19 to find maximum w-independent s-dicut family and prove the following theorem.

▶ **Theorem 20.** *There exists an algorithm that takes a directed graph $G = (V, E, w)$, $s \in V$ and outputs the tree representation $T_{\mathcal{F}}$ of the optimal w-independent s-dicut family $\mathcal{F}$ in $O(m + n \log n)$ time.*

Proof. Given a directed graph $G = (V, E, w)$, construct the graph $G' = G^0 \cup G^R = (V, A', w')$ which has $O(m)$ edges and n vertices. Give G' and $s \in V$ as inputs to the algorithm from Theorem 19 which outputs a tree representation $T_{\mathcal{F}}$ of the optimal w'-independent s-cut family $\mathcal{F}$ in $O(m + n \log n)$ time.

We prove that $\mathcal{F}$ is also an optimal w-independent s-dicut family by first proving that every s-cut $S \in \mathcal{F}$ in G' is a s-dicut in the original graph G, proving feasibility, and that the size of $\mathcal{F}$ equals the cost of an optimal s-dijoin, proving optimality.

Consider any $S \in \mathcal{F}$. Since S is an s-cut in G', the set $\delta^+_{G'}(S)$ cannot contain zero-weight edges e (from G^0) because these edges have weight $w(e) = 0$ and hence cannot appear in any cut of the w'-independent family. Thus, all edges in the s-cut belong to G^R, so

$$\delta^+_{G'}(S) = \delta^+_{G^R}(S) = \delta^-_G(S).$$

Similarly, $\delta^-_{G'}(S)$ cannot contain any edges of G^R because that would force $\delta^+_{G'}(S)$ to include a zero-weight edge, which is impossible. Hence,

$$\delta^+_G(S) = \delta^-_{G^R}(S) = \emptyset.$$

Therefore, S is a s-dicut in G. As each edge e in the s-cut is from G^R and occurs at most $w(e^R)$ times, the corresponding edge e^R in G occurs in at most $w(e^R)$ many s-dicuts making $\mathcal{F}$ a w-independent s-dicut family.

From Theorem 18, the size of the maximum w-independent s-cut family equals the cost of the minimum cost s-arborescence on G'. The former quantity equals the size of a w-independent s-dicut family as shown above and the latter equals the cost of an optimal s-dijoin by Theorem 9 proving optimality. ◀

The Algebraic Cost of a Boolean Sum

Ian Orzel
Department of Computer Science, The University of Copenhagen, Denmark

Srikanth Srinivasan
Department of Computer Science, The University of Copenhagen, Denmark

Sébastien Tavenas
Chargé de Recherche CNRS au Laboratoire LAMA de l'Université Savoie Mont Blanc, France

Amir Yehudayoff
Department of Computer Science, The University of Copenhagen, Denmark
Department of Mathematics, Technion-IIT, Haifa, Israel

Abstract

It is a well-known fact that the permanent polynomial is complete for the complexity class VNP, and it is largely suspected that the determinant does not share this property, despite its similar expression. We study the question of why the VNP-completeness proof of the permanent fails for the determinant. We isolate three fundamental properties that are sufficient to prove a polynomial sequence is VNP-hard, of which two are shared by both the permanent and the determinant. We proceed to show that the permanent satisfies the third property, which we refer to as the "cost of a boolean sum", while the determinant does not, showcasing the fundamental difference between the polynomial families. We further note that this differentiation also applies in the border complexity setting and that our results apply for counting complexity.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic Complexity, Computational Complexity, Permanent, Determinant

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.47

Funding *Ian Orzel*: funded by the European Research Council (ERC) under grant agreement no. 101125652 (ALBA).
Srikanth Srinivasan: funded by the European Research Council (ERC) under grant agreement no. 101125652 (ALBA).
Sébastien Tavenas: supported by ANR-22-CE48-0007.
Amir Yehudayoff: supported by a DNRF chair grant, and partially supported by BARC.

1 Introduction

In an attempt to introduce an algebraic approach to solving the classical P versus NP problem, [15] introduces algebraic circuits and the complexity classes VP and VNP. Central to this approach were two fundamental objects: the determinant and the permanent. The determinant belongs to the class VP (it is, in fact, complete for a restriction of this class, VBP, or even VP if we consider quasi-polynomial projections), and the permanent is complete for the class VNP, as proven in [15]. We suspect that the determinant is not VNP-complete, meaning that Valiant's proof of the VNP-hardness of the permanent should fail when the determinant is used in its place. It has been unclear exactly why this is the case, although one would expect it to be related to the construction of a particular graph-theoretical gadget, known as the *iff-coupling* gadget, being impossible in the model of the determinant. However, this is a somewhat narrow and technical reason.


© Ian Orzel, Srikanth Srinivasan, Sébastien Tavenas, and Amir Yehudayoff;
licensed under Creative Commons License CC-BY 4.0

This paper seeks to develop a more general understanding of the reasons this VNP-hardness proof fails when applied to the determinant. In exploring this question, we simplify the proof for the VNP-completeness of the permanent and isolate the exact property that the determinant lacks which causes this VNP-hardness proof to fail (assuming all other properties are the same).

Recall that the VNP-completeness proof of the permanent begins by taking a formula and summing it over all 0-1 substitutions of some of its variables (representing an element of a VNP polynomial sequence). This formula is then transformed into a similarly-sized permanent. We then iteratively remove each 0-1 substitution of a particular variable by modifying the matrix we take the permanent of, only increasing its size by a small amount. This iterative removal is the key piece of the proof that makes it work in the case of the permanent, but, to the authors' knowledge, there does not exist a proof that this step breaks down for the determinant. Studying this step in its full generality is difficult, as the given matrix could have a complex structure. In this paper, we simplify the VNP-completeness proof by showing that we can, without loss of generality, assume that the matrix has exactly two instances of the variable that we seek to remove. Using a small (standard) gadget construction, a boolean sum over two variables can be simulated by a permanent of a matrix that is bigger *only by an additive constant*, which shows that the permanent is VNP-complete. We can then show that the proof fails for the determinant by proving that the last step of this process must incur a super-constant (in fact, a constant *multiplicative factor*) increase in the size of the matrix.

We generalize this concept of "iterated substitutions" to arbitrary polynomials through the boolean sum. Given a polynomial $f \in \mathbb{F}[\mathcal{X}]$ and a subset of its variables $S \subseteq \mathcal{X}$, we can represent the boolean sum, which we denote by $\Sigma_S f$, as the sum of taking f with each variable in S set to 0 and the same with respect to 1. We can now rephrase this step of the proof as converting a boolean sum over an arbitrary-sized set of variables to multiple boolean sums over two variables.

In this paper, we have isolated the following properties as sufficient for proving the VNP-hardness of a polynomial:

1. The polynomial sequence is VF-hard, i.e., a polynomial computed by a formula can be represented as a projection of a polynomial in the sequence (we can replace arguments by variables or constants), indexed by a polynomial in the size of the formula,
2. A boolean sum over many variables can be rewritten as a sequence of boolean sums of the polynomial, each over two variables, and
3. A boolean sum of a polynomial over two variables can be rewritten as a simple projection of itself, whose index is only increased by a fixed constant.

With these properties in mind, the main results of our paper can be summarized as follows:

- A polynomial sequence satisfying properties (1), (2), and (3) is VNP-hard.
- Both the determinant and the permanent satisfy properties (1) and (2).
- The permanent satisfies property (3), but the determinant does not.

In short, we have isolated the simple property that differentiates the determinant from the permanent in our ability to prove VNP-completeness.

1.1 Related Work

The focus of this paper is on studying the noteworthy differences in properties between the permanent and the determinant polynomials, which is an important research direction, as it seems that the separation between VP and VNP is encoded in the differences between these polynomials. It is therefore that the differences in these polynomials have been well-studied.

One significant line of study was that seeking to find lower bounds on the size of a determinant that simulates the permanent (determinantal complexity). In a result of von zur Gathen [17], a lower bound was determined by studying the algebraic-geometric properties of the determinant and the permanent. Specifically, there was a difference between the dimension of the spaces of singular points of the determinant and permanent (points where a polynomial and all of its first-order partial derivatives vanish). Then, in a result of Mignon and Ressayre [11] (see also [3]), the bound was strengthened by studying the rank of the Hessian of these polynomials at vanishing points. It was shown that all zeros of the determinant have a very small Hessian rank, whereas the permanent has zero points with very large Hessian rank.

The work of Landsberg and Ressayre [8] gives an exponential lower bound on the determinantal complexity of the permanent in a restricted setting. They showed that any reductions from the permanent to the determinant that preserve the underlying symmetries must have exponential cost.

Recent work of Hrubeš and Joglekar [6] points to a distinction in reductions to *read-once* determinants and permanents, which forces each variable to appear in the corresponding matrix at most once. Here, they show that every multilinear polynomial can be represented as a read-once permanent, whereas representations using the determinant could require a variable to appear at least $\omega(\sqrt{n}/\lg n)$ times.

1.2 Our Results

We have isolated three properties that are fundamental to understanding our VNP-completeness results for the permanent. We will begin by formally stating these properties.

▶ **Definition 1.** *We say that a polynomial sequence $f = (f_i)_{i\in\mathbb{N}} \in \mathbb{F}[\mathcal{X}]$* simulates formulas *if, for every polynomial $g \in \mathbb{F}[\mathcal{X}]$ that can be computed by a formula of size s, g is a projection of some f_i, where i is polynomial in s. In other words, f is VF-hard.*

We further say that $k \in \mathbb{N}$ variable instances are sufficient for boolean sums of *f if, for every boolean sum $\Sigma_{S_1} \dots \Sigma_{S_\ell} f_i$ [see (2.1)], there is an ℓ' and m that are polynomial in i and ℓ such that this sum can be rewritten as $\Sigma_{T_1} \dots \Sigma_{T_{\ell'}} f_m$, where the size of each T_i is k.*

Finally, we define the algebraic additive cost of boolean summation over f via the map $\alpha_f : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$, defined as follows: for each $s \in \mathbb{N}$, let $\alpha_f(s)$ be the minimum integer[1] *k so that, for every n and $S \subseteq \mathcal{X}_n$ of size $|S| \leq s$, the polynomial $\Sigma_S f_n$ is a simple projection of f_{n+k}. In other words, $\alpha_f(s)$ is the additive f-cost of performing a boolean sum over s variables.*

We can further naturally move the definition of α_f into the border complexity setting, which we denote by $\underline{\alpha_f}$. Formally, we let $\underline{\alpha_f}(s)$ be the minimum integer k so that there is a border sequence $F = \overline{(F_i)}_{i\in\mathbb{N}}$ over $\mathbb{F}(\epsilon)[\mathcal{X}]$ such that, for every $i \in \mathbb{N}$, there exist polynomials $g_i \in \mathbb{F}[\mathcal{X} \cup \{\epsilon\}]$ and $Q_i \in \mathbb{F}[\epsilon]$ and $M \in \mathbb{Z}$ such that

$$F_i = \frac{\epsilon^M \cdot f_i + \epsilon^{M+1} \cdot g_i}{1 + \epsilon Q_i(\epsilon)},$$

and furthermore, $\alpha_F(s) = k$. The algebraic expression captures our intuition that $\lim_{\epsilon\to 0} F_i = f_i$.

[1] If there is no such integer, then it is infinite.

▶ Remark 2. Notice that $\alpha_f(s)$ has no dependence on n (the number of variables) and instead is a bound that applies for all n. One could modify the definition to allow for dependence on n, let's call this new function $\beta_f(s,n)$. Then, instead of the hypothesis we include in Theorem 3, namely $\alpha_f(k) < \infty$, we could rephrase it in terms of $\beta_f(s,n)$. Clearly, $\alpha_f(k) < \infty$ if and only if $\beta_f(s,n) = O(1)$, but, more generally, it is sufficient that the sequence $n, n+\beta_f(s,n), n+\beta_f(s,n+\beta_f(s,n)), \ldots$ iterated n times is polynomially bounded. Observe that a lower bound of $\beta_f(s,n) \geq (1+\epsilon)n$ is enough to make the sequence grow exponentially. To avoid the technicalities involved in characterizing such functions β_f, we have decided to use the $\alpha_f(s)$ notation.

We can now introduce our main results that use these properties. We first have that these properties are sufficient for VNP-hardness.

▶ **Theorem 3.** *If, for a fixed $k \in \mathbb{N}$, $f = (f_i)_{i\in\mathbb{N}} \in \mathbb{F}[\mathcal{X}]$ is such that*

1. *f simulates formulas,*
2. *k variables instances are sufficient for boolean sums over f, and*
3. *$\alpha_f(k) < \infty$,*

then f is VNP-hard.

Further, if the projections and reductions associated with these three properties can be completed in polynomial time, then f is #P-hard.

We use perm and det to denote the permanent and determinant polynomials respectively. It is a well-known fact that perm and det can simulate formulas.

▶ **Lemma 4** ([15])**.** *The* perm *and* det *polynomial sequences can simulate formulas.*

Our main deviation from the well-known VNP-hardness proof is through the decrease of the number of instances of the variables we sum over to two. This idea is stated in the following lemma.

▶ **Lemma 5.** *Two variables instances are sufficient for boolean sums over the* perm *and* det *polynomials.*

Finally, the main result of this paper is through differentiating the determinant from the permanent, which is stated in the following theorem.

▶ **Theorem 6.** *We have that $\alpha_{\mathsf{perm}}(2) \leq 3$, but $\alpha_{\mathsf{det}}(2) = \infty$. We further observe that $\underline{\alpha_{\mathsf{det}}}(2) = \infty$.*

1.3 Structure of Paper

First, in Section 2, we will introduce our notation and definitions. Then, in Section 3, we will prove Theorem 6, which will differentiate the permanent from the determinant. We will do this using a well-known result of Mignon and Ressayre[11] and through a careful selection of matrices. In Section 4, we will prove Lemma 5, showing that we can transform arbitrarily-sized boolean sums to those of size two. In Section 5, we provide the proof of Theorem 3, which is (in our opinion) a slightly simpler and more modular proof of the VNP-completeness of the permanent.

2 Preliminaries

Let $\mathbb{F}$ be a field of characteristic not two. Let $\mathcal{X}$ be an infinite set of indeterminants and $\mathbb{F}[\mathcal{X}]$ be the polynomial ring containing polynomials that use a finite number of indeterminants in $\mathcal{X}$. We will write $(f_i)_{i\in\mathbb{N}} \in \mathbb{F}[\mathcal{X}]$ to denote a sequence a polynomials (we will somtimes simply write it as (f_i)), and it is assumed that $f_i \in \mathbb{F}[\mathcal{X}_i]$, where $\mathcal{X}_i \subseteq \mathcal{X}$ has size polynomial in i. Given a set S, we may use $S^{\mathcal{X}_i}$ to denote an assignment of the variables in $\mathcal{X}_i$ to elements of S, so that we can then see $f_i(a)$ to represent evaluating f_i at the point $a \in S^{\mathcal{X}_i}$.

We will use the typical definitions for circuits, formulas, the complexity classes VF, VP, and VNP, and hardness within these classes. For more information, see [15].

A fundamental tool in computational complexity theory is reduction between problems [7]. Reductions allow us to compare the complexities of different problems. In the algebraic setting, reductions are projections. The polynomial $f \in \mathbb{F}[\mathcal{X}]$ is a projection of $g \in \mathbb{F}[\mathcal{Y}]$ if there is a map $\phi : \mathcal{Y} \to \mathcal{X} \cup \mathbb{F}$ so that $f = \lambda(g \circ \phi)$, where $\lambda \in \mathbb{F} \setminus \{0\}$. The projection is called *simple* if for each $x \in \mathcal{X}$ the cardinality of $\phi^{-1}(x)$ is at most one.

For boolean summation, we use the following notation. For $S \subseteq \mathcal{X}$, denote by f_S the polynomial in the variables $\mathcal{X} \cup \{y\}$, where y is a new variable, that is obtained from f by the substitution $x = y$ for all $x \in S$. For example, $(x_1 + x_2^2 + x_3)_{\{x_1,x_2\}} = y + y^2 + x_3$. Denote by $\Sigma_S f$ the polynomial

$$\Sigma_S f = f_S|_{y=0} + f_S|_{y=1}. \tag{2.1}$$

For example, $\Sigma_{\{x_1,x_2\}}(x_1 + x_2^2 + x_3) = 2 + 2x_3$. The variables in S do not appear in f_S and $\Sigma_S f$.

For two polynomial sequences $g = (g_i)_{i\in\mathbb{N}}$ and $f = (f_i)_{i\in\mathbb{N}}$, we say that g is an iterated boolean sum of f if, for every $i \in \mathbb{N}$, there are disjoint sets $S_1, \ldots, S_\ell \subseteq \mathcal{X}$ such that $g_i = \Sigma_{S_1} \ldots \Sigma_{S_\ell} f_j$, where ℓ and j are polynomial in i. Observe that a polynomial sequence being in VNP is equivalent to there being a sequence in VF each element in the VNP sequence is an iterated boolean sum of a (polynomially-indexed) element in the VF sequence.

The two central polynomial families in algebraic complexity are the permanent and the determinant, defined by,

$$\mathsf{perm}(X) = \mathsf{perm}_n(X) = \sum_{\pi} \prod_{i} x_{i,\pi(i)}$$

and

$$\mathsf{det}(X) = \mathsf{det}_n(X) = \sum_{\pi} (-1)^{\mathsf{sign}(\pi)} \prod_{i} x_{i,\pi(i)},$$

where $X = (x_{i,j})$ is an $n \times n$ matrix of variables, the sum is over permutations π of $[n]$, and the product is over $i \in [n]$. The permanent and determinant can be equivalently defined as the sum of the (signed) weights of cycle covers over a graph, whose adjacency matrix is given; see [9]. The permanent is VNP-complete, so that VP $\neq$ VNP if and only if perm is not in VP [15]. It also appears in many places in computer science because it encapsulates many counting problems (see e.g. [16, 14, 1] and references within); by definition, it counts the number of perfect matchings in a bipartite graph. The VP versus VNP problem is (essentially) about the relationship between the permanent and the determinant (see [15, 12, 10] and references within).

The determinant defines a complexity measure; the determinantal complexity $\mathsf{dc}(f)$ of a polynomial f is the minimum k so that f is a projection of det_k. It is known that $\mathsf{dc}(f) < \infty$ for all f. The VP versus VNP problem is roughly captured by *what is* $\mathsf{dc}(\mathsf{perm}_n)$? Proving that $\mathsf{dc}(\mathsf{perm}_n)$ is super-polynomial in n is one of the major open problems in computer science. The best lower bound we know is $\mathsf{dc}(\mathsf{perm}_n) \geq \frac{n^2}{2}$; see [11, 3].

Given a complexity measure $L : \mathbb{F}[\mathcal{X}] \to \mathbb{N}$, we can define its corresponding "border" measure, denoted $\underline{L}$, to be the polynomial with the smallest L measure that approximates the original polynomial. More formally, given an $f \in \mathbb{F}[\mathcal{X}]$, we define $\underline{L}(f)$ to be the smallest k such that there exist $F \in \mathbb{F}(\epsilon)[\mathcal{X}]$, $g \in \mathbb{F}[\mathcal{X} \cup \{\epsilon\}]$, $Q \in \mathbb{F}[\epsilon]$, and $M \in \mathbb{Z}$ such that

$$F = \frac{\epsilon^M f + \epsilon^{M+1} g}{1 + \epsilon Q(\epsilon)},$$

and furthermore, $L(F) = k$.

In the field of counting complexity, we study the complexity of functions that "count" some quantity, mapping $\varphi : \{0,1\}^* \to \mathbb{N}$. Recall that we define the class NP as the set of decision problems, denoted by $\varphi : \{0,1\}^* \to \{0,1\}$, such that there is a Turing Machine taking in the input and a special "witness" string such that the Turing Machine runs in polynomial time and a string is in the language if and only if there is at least one "witness" string for which the Turing Machine accepts it. We then define the complexity class ♯P similarly, but instead we require the related Turing Machine to have the number of accepting witness strings to a given input to be the same as that defined by φ. The quintessential ♯P-complete problem is ♯3-SAT, which, given a 3-CNF formula, counts the number of satisfying assignments.

3 Cost of Boolean Summation

This section will focus on proving Theorem 6. We will separate the proof of this theorem into two sub-claims. The first is used to show that $\alpha_{\mathsf{det}}(2) = \infty$.

▷ Claim 7. For any $n \in \mathbb{N}$, we have that

$$\mathsf{dc}(\Sigma_{\{x_{1,1},x_{2,2}\}}\mathsf{det}_n) \geq 2(n-2).$$

Next, we will prove that $\alpha_{\mathsf{perm}}(2) \leq 3$ through another claim.

▷ Claim 8. There exists a matrix $E \in (\mathcal{X} \cup \mathbb{F})^{(n+3)\times(n+3)}$ such that

$$\Sigma_{\{x_{1,1},x_{1,2}\}}\mathsf{perm} = \mathsf{perm}(E).$$

Observe that, due to row and column swaps (and the fact that, if the two indeterminants were on the same row or column, the claim would be trivially true), these claims imply what we want.

3.1 Lower Bound on Determinantal Boolean Sums

We now prove that $\alpha_{\mathsf{det}}(2) = \infty$ by proving Claim 7. The proof relies on a mechanism developed by Mignon and Ressayre to lower bound the determinantal complexity [11]. For a polynomial $f \in \mathbb{F}[\mathcal{X}]$, denote by H_f the $|\mathcal{X}| \times |\mathcal{X}|$ Hessian matrix of f defined by

$$(H_f)_{x,x'} = \frac{\partial^2}{\partial x \partial x'} f.$$

▶ **Lemma 9** ([11]; see also Lemma 13.3 in [4]). *Let f be a polynomial in the variables $x = (x_1, \ldots, x_n)$ and let $a \in \mathbb{F}^n$ be so that $f(a) = 0$ then*

$$\mathsf{dc}(f) \geq \frac{\mathsf{rank}(H_f|_{x=a})}{2}.$$

The theorem follows by considering the polynomial

$$f = \Sigma_{\{x_{1,1},x_{2,2}\}}\mathsf{det}_n$$

and the $n \times n$ matrix

$$A = \begin{bmatrix} & \frac{1}{2} & \\ 1 & & \\ & & I \end{bmatrix},$$

where I is the $(n-2) \times (n-2)$ identity matrix. In the two claims below (which complete the proof), let $H = H_f$.

▷ Claim 10. $f(A) = 0$.

Proof of Claim 10.

$$f(A) = \mathsf{det}\left(\begin{bmatrix} & \frac{1}{2} & \\ 1 & & \\ & & I \end{bmatrix}\right) + \mathsf{det}\left(\begin{bmatrix} 1 & \frac{1}{2} & \\ 1 & 1 & \\ & & I \end{bmatrix}\right)$$
$$= -\frac{1}{2} + 1 - \frac{1}{2} = 0.$$ ◁

▷ Claim 11. $\mathsf{rank}(H|_{X=A}) \geq 4(n-2)$.

Proof of Claim 11. Focus on the $4(n-2)$ variables $\mathcal{V}_i = \{x_{1,i}, x_{2,i}, x_{i,1}, x_{i,2}\}$ for $i > 2$. Furthermore, we consider G to be the $4(n-2) \times 4(n-2)$ sub-matrix of H whose rows and columns are labeled by variables in $\bigcup_{i>2} \mathcal{V}_i$. Let $f_0 = \mathsf{det}_n|_{x_{1,1}=x_{2,2}=0}$ and $f_1 = \mathsf{det}_n|_{x_{1,1}=x_{2,2}=1}$, so that $f = f_0 + f_1$.

An entry in the Hessian of det is plus/minus the determinant of the $(n-2)$-minor obtained by deleting the corresponding rows and columns from X (or zero if the two variables share a row or a column).

For convenience, denote by $H_{k\ell mp}$ the $(x_{k,\ell}, x_{m,p})$-entry in H (and in G).

We first claim that after substituting A, the sub-matrix G of H has $n-2$ blocks, each of size 4×4:

$$\begin{bmatrix} G_3 & & & \\ & G_4 & & \\ & & \ddots & \\ & & & G_n \end{bmatrix}$$

where for $i > 2$

$$G_i = \begin{bmatrix} H_{1i1i} & H_{1i2i} & H_{1ii1} & H_{1ii2} \\ H_{2i1i} & H_{2i2i} & H_{2ii1} & H_{2ii2} \\ H_{i11i} & H_{i12i} & H_{i1i1} & H_{i1i2} \\ H_{i21i} & H_{i22i} & H_{i2i1} & H_{i2i2} \end{bmatrix}.$$

Indeed, for $i \neq j$, the matrix obtained from A by deleting row i and column j has a zero row so its determinant is zero. It follows for $k, \ell \in \{1, 2\}$ that

$$(H_f)_{ik\ell j}|_{X=A} = (H_{f_0})_{ik\ell j}|_{X=A} + (H_{f_1})_{ik\ell j}|_{X=A} = 0 + 0 = 0.$$

A similar argument holds for the other entries $H_{ikj\ell}$, $H_{ki\ell j}$, and $H_{kij\ell}$.

Now fix i and focus on G_i. Entries of the form $1i1i$ and $1i2i$ are also zero. So, we are left with

$$G_i = \begin{bmatrix} & & H_{1ii1} & H_{1ii2} \\ & & H_{2ii1} & H_{2ii2} \\ H_{i11i} & H_{i12i} & & \\ H_{i21i} & H_{i22i} & & \end{bmatrix}.$$

The matrix G_i is symmetric so it is enough to understand

$$\begin{bmatrix} H_{1ii1} & H_{1ii2} \\ H_{2ii1} & H_{2ii2} \end{bmatrix}.$$

This matrix is a sum of two matrices (one from f_0 and one from f_1). For f_0, this matrix is

$$\begin{bmatrix} & 1 \\ \frac{1}{2} & \end{bmatrix}.$$

For f_1, this matrix is

$$\begin{bmatrix} -1 & 1 \\ \frac{1}{2} & -1 \end{bmatrix}.$$

The sum of the two matrices is

$$\begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix},$$

which always has rank two. Rolling back

$$\mathsf{rank}\left(\begin{bmatrix} G_3 & & & \\ & G_4 & & \\ & & \ddots & \\ & & & G_n \end{bmatrix}\right) = 4(n-2). \qquad \lhd$$

▶ **Observation 12.** *We note that this result also applies in the border complexity setting. Specifically, it also follows that* $\underline{\mathsf{dc}}(\Sigma_{\{x_{1,1},x_{2,2}\}}\mathsf{det}_n) \geq 2(n-2)$. *This fact follows from [5], where it is shown that the lower bound from [11] of the determinantal complexity of the permanent also applies in the border complexity setting. Although the proof presented there is specifically for the permanent polynomial, it is easy to see that it also works if you use any arbitrary homogeneous polynomial. We should observe that* $\Sigma_{\{x_{1,1},x_{2,2}\}}\mathsf{det}_n$ *is not homogeneous, but one can easily see that we can apply the result to the natural homogenization of the polynomial and get the desired result.*

3.2 Upper Bound on Permanental Boolean Sum

In this section, we will prove that $\alpha_{\mathsf{perm}}(2) \leq 3$ by proving Claim 8. Specifically, we will select the matrix E to be

$$E = \left[\begin{array}{ccc|cccc} 1 & 1 & 1 & & & & \\ 1 & 0 & -1 & 1 & & & \\ -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} & & 1 & & \\ \hline & 1 & & 0 & x_{1,2} & x_{1,3} & \cdots \\ & & 1 & x_{2,1} & 0 & x_{2,3} & \\ & & & x_{3,1} & x_{3,2} & x_{3,3} & \\ & & & \vdots & & & \ddots \end{array}\right].$$

Proof of Claim 8. To see this, denote by $X[b]$ the matrix X after the substitution $x_{1,1} = x_{2,2} = b$ for $b \in \{0,1\}$. For $S \subseteq \{1,2\}$, denote by $X[b]_{-S}$ the matrix $X[b]$ after deleting the rows and columns in S. We have

$$\mathsf{perm}(X[1]) = \mathsf{perm}\left(\begin{bmatrix} 1 & x_{1,2} & x_{1,3} & \\ x_{2,1} & 1 & x_{2,3} & \\ x_{3,1} & x_{3,2} & x_{3,3} & \\ & & & \ddots \end{bmatrix}\right)$$
$$= \mathsf{perm}(X[0]) + \mathsf{perm}(X[0]_{-\{1\}}) + \mathsf{perm}(X[0]_{-\{2\}}) + \mathsf{perm}(X[0]_{-\{1,2\}}).$$

Now, consider a permutation π such that the associated value of the permutation on E is nonzero. We can then notice that such a permutation falls into one of four categories:

- $\pi(\{1,2,3\}) = \{1,2,3\}$,
- $\pi(2) = 4$ and $\pi(\{1,3\}) = \{1,3\}$,
- $\pi(3) = 5$ and $\pi(\{1,2\}) = \{1,2\}$, or
- $\pi(2) = 4$, $\pi(3) = 5$, and $\pi(1) = 1$.

Let U be the 3×3 upper-left block

$$U = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \end{bmatrix}.$$

For $S, T \subseteq \{1,2,3\}$ with $|S| = |T|$, we denote by U_S^T the sub-matrix of U obtained by keeping the rows of S and the columns of T. The matrix U is chosen such that

$$\mathsf{perm}(U^{\{1,3\}}_{\{1,2\}}) = \mathsf{perm}(U^{\{1,2\}}_{\{1,3\}}) = 0,$$
$$\mathsf{perm}(U^{\{1\}}_{\{1\}}) = \mathsf{perm}(U^{\{1,2\}}_{\{1,2\}}) = \mathsf{perm}(U^{\{1,3\}}_{\{1,3\}}) = 1,$$
$$\mathsf{perm}(U) = 2.$$

In the first of the above 4 cases, it must be that $\pi(\{4,5,\ldots,m\}) = \{4,5,\ldots,m\}$. Therefore, the sum of the weights of these permutations corresponds to

$$\mathsf{perm}\left(\left[\begin{array}{c|c} U & \\ \hline & X[0] \end{array}\right]\right) = 2\mathsf{perm}(X[0]).$$

In the second case, it must be that $\pi(4) = 2$ and $\pi(\{5,6,\ldots,m\}) = \{5,6,\ldots,m\}$. Thus, this case corresponds to

$$\mathsf{perm}\left(\left[\begin{array}{c|c} U^{\{1,3\}}_{\{1,3\}} & \\ \hline & X[0]_{-\{1\}} \end{array}\right]\right) = \mathsf{perm}(X[0]_{-\{1\}}).$$

In the third case, we have that $\pi(5) = 3$ and $\pi(\{4, 6, 7, \dots, m\}) = \{4, 6, 7, \dots, m\}$. Thus, this case corresponds to

$$\mathsf{perm}\left(\left[\begin{array}{c|c} U^{\{1,2\}}_{\{1,2\}} & \\ \hline & X[0]_{-\{2\}} \end{array}\right]\right) = \mathsf{perm}(X[0]_{-\{2\}}).$$

Finally, in the fourth case, it must be that $\pi(4), \pi(5) \in \{2, 3\}$ and $\pi(\{6, 7, \dots, m\}) = \{6, 7, \dots, m\}$. This case then corresponds to

$$\mathsf{perm}\left(\left[\begin{array}{c|c} U^{\{1\}}_{\{1\}} & \\ \hline & X[0]_{-\{1,2\}} \end{array}\right]\right) = \mathsf{perm}(X[0]_{-\{1,2\}}).$$

Summing it together,

$$\mathsf{perm}(E) = \mathsf{perm}(X[0]) + \mathsf{perm}(X[1]).$$ ◁

3.3 Boolean Sum over a Single Variable

Interestingly, considering boolean sums of two variables is necessary to separate perm from det. This is becaues the boolean cost of removing one variable is the same for both polynomial sequences.

▷ Claim 13. We have that $\alpha_{\mathsf{det}}(1) = \alpha_{\mathsf{perm}}(1) = 0$.

Proof.

$$\begin{aligned}
&\Sigma_{\{x_{1,1}\}}\mathsf{det}(X) \\
&= \mathsf{det}\begin{bmatrix} 0 & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix} + \mathsf{det}\begin{bmatrix} 1 & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix} \\
&= 2\mathsf{det}\begin{bmatrix} 1/2 & x_{1,2} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,n} \end{bmatrix}.
\end{aligned}$$

A similar calculation holds for the permanent. ◁

4 Two variables suffice for VNP-hardness

The aim of this section is to prove Lemma 5 that shows that summation over two variables suffices for VNP-hardness. We prove it for the determinant (the proof for the permanent is similar). The following construction shows how to replace a boolean summation over m variables by m boolean summations over two variables.

▶ **Observation 14.** *Let $f \in \mathbb{F}[\mathcal{X}]$ and $S = \{x_1, \dots, x_m\} \subseteq \mathcal{X}$. Let $y_1, \dots, y_m$ be m new variables, and define*

$$g(y) = \Big(\prod_{i\in[m]} y_i\Big) + \Big(\prod_{i\in[m]} (1 - y_i)\Big).$$

Then

$$\Sigma_{\{x_1,y_1\}}\Sigma_{\{x_2,y_2\}}\dots\Sigma_{\{x_m,y_m\}}gf = \Sigma_S f.$$

Indeed, on the l.h.s. there is a sum over 2^m terms, but g picks exactly two of them that perfectly agree with the r.h.s.

The observation allows to replace a boolean sum over many variables by several sums, each over just two.

▶ **Observation 15.** *Let g be the polynomial defined above. Then, there is a $2m \times 2m$ matrix A with entries in $\{y_1,\dots,y_m,0,1,-1\}$ so that each y_i appears once in A and*

$$g = \mathsf{det}(A).$$

The matrix A is the m-variate version of the following matrix:

$$A = \left[\begin{array}{cc|cc|cc|cc} -1 & 1 & -1 & & & & & \\ -1 & y_1 & & & & & & \\ \hline & & -1 & 1 & -1 & & & \\ & & -1 & y_2 & & & & \\ \hline & & & & -1 & 1 & -1 & \\ & & & & -1 & y_3 & & \\ \hline 1 & & & & & & -1 & 1 \\ & & & & & & -1 & y_4 \end{array}\right].$$

To see that $\mathsf{det}(A)$ is the polynomial g, it is easier to see the determinant as a signed sum of the weights of the cycles covers of the graph G_A whose A is the adjacency matrix (see Figure 1)

$$\mathsf{det}(A) = \sum_{C \text{ cycles cover}} \left[\prod_{\sigma \text{ cycle of } C} (-1)^{1+\text{length of } C}\text{weight(C)}\right].$$

The reader could find more details about this characterization by cycles covers in [9]. The only cycle which contains a dashed edge is exactly the union of the dashed edges. So the only cover that contains this cycle also contains the m y_i-loops. It corresponds to one cover of signed weight $\left((-1)^{m+1}(-1)^{m-1}\right)\prod_{i=1}^m y_i = \prod_{i=1}^m y_i$. Otherwise, the covers contain no dashed edges and are given by m disjoint graphs. The signed weight is the product of them: $\prod_i((-1)(-1)+(1)(-y_i)) = \prod_i(1-y_i)$. The sum of the covers gives exactly the polynomial g.

▶ Remark 16. For the permanent, every -1 above the diagonal of the matrix A should be changed into a 1, and the 2×2 blocks into $\begin{bmatrix} -1 & 1 \\ 1 & y_1 \end{bmatrix}$.

Proof of Lemma 5. Let $B \in (\mathcal{X}\cup\mathbb{F})^{n\times n}$ and $S_1,\dots,S_\ell \subseteq \mathcal{X}$ be such that $S_1,\dots,S_\ell$ are disjoint. Then, by previous

$$\Sigma_{S_\ell}\mathsf{det}(B) = \Sigma_{\{x_1,y_1\}}\dots\Sigma_{\{x_m,y_m\}}g\mathsf{det}(B) = \Sigma_{\{x_1,y_1\}}\dots\Sigma_{\{x_m,y_m\}}\mathsf{det}\begin{pmatrix} B & 0 \\ 0 & A \end{pmatrix}$$

Notice here that $m \le |S_\ell|$ and the size of the block matrix is at most $n+2m \le n+2|S_\ell|$. We can then use induction on $S_1,\dots,S_{\ell-1}$ to conclude that we can write

$$\Sigma_{S_1}\dots\Sigma_{S_\ell}\mathsf{det}(A) = \Sigma_{T_1}\dots\Sigma_{T_{\ell'}}\mathsf{det}(C),$$

where $\ell' \le \sum_{i=1}^{\ell}|S_\ell| \le n^2$ and the size of C is at most $n+2\sum_{i=1}^{\ell}|S_\ell| \le n+2n^2$. ◀

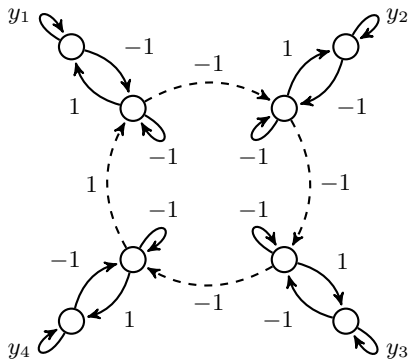


Figure 1 The graph G_A.

5 The permanent is VNP-complete

This section will finally provide the proof of Theorem 3, showing that the three properties we have isolated are sufficient for VNP-hardness. The proof proceeds similarly to Valiant's VNP-completeness proof of the permanent with a few variations. This section thus also provides a (arguably) more modular proof of the VNP-completeness of the permanent.

Proof of Theorem 3. Because f simulates formulas, we can write every n-variate $g \in \mathbb{F}[\mathcal{Y}]$ in VNP (see [2, 13]) as

$$g = (\Sigma_{S_1} \dots \Sigma_{S_\ell} f_d)(a)$$

where $a \in (\mathbb{F} \cup \mathcal{Y})^{\mathcal{X}_d}$ is a variable assignment such that d is polynomial in n, each entry in a is a variable or a field element, and $S_1, \dots, S_\ell$ are pairwise disjoint (implying ℓ is polynomial in d).

Because k variables are sufficient for boolean sums over f, we can assume without loss of generality that $|S_1| = \dots = |S_\ell| = k$. Further, as $\alpha_f(k) < \infty$, we know that we can write

$$\Sigma_{S_\ell} f_d = f_{d+\alpha_f(k)}(b),$$

for some simple variable assignment $b \in (\mathbb{F} \cup \mathcal{Y})^{\mathcal{X}_{i+\alpha_f(k)}}$. Observe that, because this is a simple projection, we can easily (after modifying our sets $S_1, \dots, S_{\ell-1}$) bring out b and write $(\Sigma_{S_1} \dots \Sigma_{S_{\ell-1}} f_{d+\alpha_f(k)})(b') = g$, for some assignment b'. Thus, after repeating this process ℓ times, we can write

$$(\Sigma_{S_1} \dots \Sigma_{S_\ell} f_d)(a) = f_{d+\ell\alpha_f(k)}(c),$$

where $c \in (\mathbb{F} \cup \mathcal{Y})^{\mathcal{X}_{d+\ell\alpha_f(k)}}$ is a variable assignment. Because ℓ and d are polynomial in n, we can conclude that f is VNP-hard. ◀

▶ **Observation 17.** *We will then conclude with the observation that this proof also works for proving ♯P-hardness if each of the steps that we took in this proof can be done in polynomial time. Consider ♯3-SAT, a complete problem for ♯P, where we are given a 3-CNF and must return the number of satisfying assignments it has. Consider some 3-CNF φ in n variables, and observe that the number of possible clauses is polynomial in n, so we can easily encode it using one bit for each possible clause. We will write this number as m_n, and one can easily see that there is a polynomial $f_n \in \mathbb{F}[y_1, \dots, y_{m_n}, x_1, \dots, x_n]$ such that, if we provide*

the y_i variables with the 0/1 encoding of φ and the x_i variables with a variable assignment, it will output 0/1 depending on whether this assignment is a satisfying assignment of φ. One can easily see that f_n can be computed by a formula that is polynomial in n. Finally, we observe that the polynomial $g_n = \Sigma_{\{x_1\}} \dots \Sigma_{\{x_n\}} f_n$ provides the exact answer for ♯3SAT when given the binary encoding of a 3-CNF. Hence, we can carry out our previous proof from the sequence $g = (g_n)_{n \in \mathbb{N}}$ to reduce it to a polynomial with the three properties (ensuring that the steps along the way can be done in polynomial time). At the end, we conclude that we can reduce ♯3-SAT to our polynomial sequence.

References

1 Scott Aaronson. A linear-optical proof that the permanent is #P-hard. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2136):3393–3405, 2011.

2 Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Science & Business Media, 2000.

3 Jin-Yi Cai, Xi Chen, and Dong Li. Quadratic lower bound for permanent vs. determinant in any characteristic. *computational complexity*, 19(1):37–56, 2010. doi:10.1007/S00037-009-0284-2.

4 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial derivatives in arithmetic complexity and beyond. *Foundations and Trends in Theoretical Computer Science*, 6(1–2):1–138, 2011. doi:10.1561/0400000043.

5 Joshua A. Grochow. Unifying known lower bounds via geometric complexity theory. *computational complexity*, 24:393–475, 2015. doi:10.1007/S00037-015-0103-X.

6 Pavel Hrubes and Pushkar S. Joglekar. On read-k projections of the determinant. *Electron. Colloquium Comput. Complex.*, TR24-125, 2024. URL: https://eccc.weizmann.ac.il/report/2024/125.

7 Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975. doi:10.1002/NET.1975.5.1.45.

8 Joseph M. Landsberg and Nicolas Ressayre. Permanent v. determinant: An exponential lower bound assuming symmetry. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 29–35. ACM, 2016. doi:10.1145/2840728.2840735.

9 Meena Mahajan and V Vinay. Determinant: Old algorithms, new insights. *SIAM journal on Discrete Mathematics*, 12(4):474–490, 1999. doi:10.1137/S0895480198338827.

10 Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *Journal of complexity*, 24(1):16–38, 2008. doi:10.1016/J.JCO.2006.09.006.

11 Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, (79):4241–4253, 2004.

12 Ketan D Mulmuley. On P vs. NP and geometric complexity theory. *Journal of the ACM*, 58(2):1–26, 2011.

13 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3–4):207–388, 2010. doi:10.1561/0400000039.

14 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.

15 Leslie G. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.

16 Leslie G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

17 Joachim von zur Gathen. Permanent and determinant. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 398–401. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.42.

On the Roots of Independence Polynomial: Quantifying the Gap

Om Prakash
The Institute of Mathematical Sciences, HBNI, Chennai, India

Vikram Sharma
The Institute of Mathematical Sciences, HBNI, Chennai, India

Abstract

The independence polynomial of a graph G is the generating polynomial corresponding to its independent sets of different sizes. More formally, if $a_k(G)$ denotes the number of independent sets of G of size k then $I(G,z) := \sum_k (-1)^k a_k(G) z^k$. The study of evaluating $I(G,z)$ has several deep connections to problems in combinatorics, complexity theory and statistical physics. Consequently, the roots of the independence polynomial have been studied in detail. In particular, many works have provided regions in the complex plane that are devoid of any roots of the polynomial. One of the first such results showed a lower bound on the absolute value of the smallest root $\beta(G)$ of the polynomial. Furthermore, when G is connected, Goldwurm and Santini established that $\beta(G)$ is a simple real root of $I(G,z)$ smaller than one. An alternative proof was given by Csikvári. Both proofs do not provide a gap from $\beta(G)$ to the smallest absolute value amongst all the other roots of $I(G,z)$. In this paper, we quantify this gap.

2012 ACM Subject Classification Mathematics of computing → Generating functions

Keywords and phrases Independence Polynomial, Root separation, Zero-free regions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.48

Acknowledgements We would like to thank the anonymous referees for their valuable inputs.

1 Introduction

Let $G = (V, E)$ be a simple undirected graph, that is, without loops and multiple edges, with V representing its set of n vertices and E its set of edges. An **independent set** of G is a subset of vertices of V such that there is no edge between any pair of vertices in the subset. Let $a_k(G)$ denote the number of independent sets of size k in G, where $a_0(G) := 1$. Following the convention in [3], we define the **independence polynomial**

$$I(G,z) := \sum_{k=0}^{n} (-1)^k a_k(G) z^k. \tag{1}$$

Let $\beta(G)$ denote the smallest real root of $I(G,z)$. It is well known that such a root exists and is indeed in the interval $(0,1]$. Moreover, it is also known that any other root ρ of $I(G,z)$ is strictly greater than $\beta(G)$ in absolute value [5, 3]. We next describe their proofs in brief, but for this we need to introduce some notation.

For a vertex $v \in V$, let $N_G(v)$ denote the set of neighbors of v and $d(v)$ its degree in G; we will also use $N(v)$ if G is clear from the context. The set of **closed neighbors** of v is $N[v] := N(v) \cup \{v\}$. Given a subset $S \subset V$, the graph $G \setminus S$ denotes the subgraph of G induced by the vertices $V \setminus S$. A key recursive property of the independence polynomial is the following: For all vertices $u \in V$, we have

$$I(G,z) = I(G \setminus u, z) - z I(G \setminus N[u], z). \tag{2}$$


© Om Prakash and Vikram Sharma;
licensed under Creative Commons License CC-BY 4.0

One of the key tools used in both [5] and [3] is the Taylor series expansion of $1/I(G,z)$ around the origin. In [5], the vertices of the graph are treated as symbols of an alphabet and the edge relations as congruence relations on the alphabet, that is, if two vertices have an edge then the corresponding alphabets can be swapped in any string. The congruence relations impose an equivalence relation, called the trace monoid, on the set of all finite strings over the alphabet. It is well known that the nth coefficient of the power series $1/I(G,z)$ is the number of traces of length n in the monoid. Using the properties of the trace monoid, it is shown that the power series can be expressed as a rational function where both the numerator and denominator correspond to the characteristic polynomial of two positive matrices. Furthermore, the positive matrix that appears in the denominator dominates the one in the numerator entry wise. Therefore, its largest eigenvalue is unique (due to Perron-Frobenius) and it does not appear as an eigenvalue of the numerator. This establishes the uniqueness of the pole, which is also $\beta(G)$. We do not see an immediate way to quantify the proof.

In [3] it is shown, using Equation (1), that the coefficients of the power series $1/I(G,z)$ are all positive. Moreover, from Pringsheim's theorem we know that the radius of convergence of this power series is $\beta(G)$. Now consider the power series $1/I(H,z)$ for any *proper subgraph* H of G. We can express

$$\frac{1}{I(G,z)} = \frac{I(H,z)}{I(G,z)}\frac{1}{I(H,z)}.$$

By repeated applications of Equation (1), along with induction, one can argue that the coefficients of both $I(H,z)/I(G,z)$ and $1/I(H,z)$ are positive. Therefore, the coefficients of the series on the left-hand side above are greater than the coefficients of $1/I(H,z)$. Hence, $\beta(G) < \beta(H)$. By an inductive argument, it is then shown that $\beta(G)$ is in fact a simple root of $I(G,z)$, when G is connected. To show that any other root ρ of $I(G,z)$ is strictly greater than $\beta(G)$ in absolute value, they consider the function

$$f_u(z) := \frac{zI(G \setminus N[u], z)}{I(G \setminus u, z)},$$

where $u \in V$. An inductive argument, similar to the one used for $1/I(G,z)$, shows that the coefficients (except the constant coefficient) of this power series are also positive. Now if ρ is any other root of $I(G,z)$ with absolute value $\beta(G)$ then $f_u(\rho) = f_u(|\rho|) = 1$, that is, f_u is periodic on the circle $\beta(G)e^{i\theta}$, where $\theta \in [0, 2\pi)$ and $i := \sqrt{-1}$. Using the "Daffodil Lemma" from complex analysis [4, p. 266], this implies that the coefficients of $f_u(z)$ are a subset of an arithmetic progression. However, since the kth coefficient of $f_u(z)$ asymptotically is of the form $(\beta(G \setminus u))^{-k}$, it does not satisfy an arithmetic progression and this gives us a contradiction that ρ can have the same absolute value as $\beta(G)$. As the proof is by contradiction, it also fails to quantify the gap between $\beta(G)$ and ρ.

The insight in this paper builds on the observation that since $f_u(z)$ is holomorphic for all z such that $|z| < \beta(G \setminus u)$, from the Maximum Modulus Principle [4, p. 545] we know that the largest absolute value of $f_u(z)$ on the disc $D(0, \beta(G)e^{i\theta})$ is attained on its boundary. Moreover, as the coefficients of $f_u(z)$ are positive, a simple calculation shows that the maximum absolute value is attained on the positive real axis, namely at $\beta(G)$. This gives an alternate proof to the ones given above. To quantify this argument, we proceed in two steps: first, we show that the function is univalent in a neighborhood of $\beta(G)$, and second, by constructing a disc around all points with absolute value $\beta(G)$, except in a certain neighborhood of $\beta(G)$, where $f_u(z)$ does not take the value one. The two steps should

intuitively hold, since in the case of the former, as the derivative $f'_u(z)$ does not vanish at $\beta(G)$ there must be a neighborhood of $\beta(G)$ where the function is injective; the latter case follows from the continuity of $f_u(z)$. The main challenge is to quantify these two intuitive ideas. For this purpose, we need tools, such as, Smale's γ-function to study the function locally, and some simple results from complex analysis on radius of univalence of a function, such as $f_u(z)$. The main result of the paper is the following:

▶ **Theorem 1.1.** *Let G be a connected graph on n vertices. Then the disc centered at the origin*

$$D\left(0, \beta(G) + \left(\frac{\beta(G)}{n}\right)^{O(n)}\right)$$

contains only the smallest root $\beta(G)$ of $I(G, z)$.

In the next section, we describe the main results of the paper with intuitive details. The proofs of these results are developed in the subsequent sections. The necessary preliminary results and definitions from graph theory and complex analysis that are needed are provided in the appendix Appendix A.

2 Main Results

Our focus will be to understand the properties of the following function: For any $u \in V$, let

$$f_u(z) := \frac{zI(G \setminus N[u], z)}{I(G \setminus u, z)}, \tag{3}$$

For example, consider the star graph S_n with one central vertex of degree n connected to n leaves. It is not hard to verify that its independence polynomial is $(1-z)^n - z$. Now, if u is the "center" vertex in S_n then $f_u(z) = z/(1-z)^n$, but if u is one of the leaves in S_n then

$$f_u(z) = z(1-z)^{n-1}/((1-z)^{n-1} - z). \tag{4}$$

If $N[u] = \{u, u_1, \ldots, u_k\}$, where $k = d(u)$, then

$$f_u(z) = z\frac{I(G \setminus \{u, u_1\}, z)}{I(G \setminus u, z)} \cdots \frac{I(G \setminus \{u, u_1, \ldots, u_k\}, z)}{I(G \setminus \{u, u_1, \ldots, u_{k-1}\}, z)}.$$

By k applications of Equation (2), one can recursively construct functions $g_j(z)$, $j = 1, \ldots, \ell$ such that

$$f_u(z) = \frac{z}{(1-z)^{d(v)-\ell}\prod_{j=1}^{\ell}(1 - g_j(z))}, \tag{5}$$

where $g_j(z)$ is not identity for all j. Again consider S_n with u as one of the leaves then the function given in Equation (4) can be re-written as

$$f_u(z) = \frac{z}{1 - \frac{z}{(1-z)^{n-1}}}.$$

The **depth of** $f_u(z)$ is one more than the maximum depth of $g_j(z)$'s, where the base case

$$f_u(z) = \frac{z}{(1-z)^{\ell}}, \tag{6}$$

for $\ell \geq 1$, has depth one. The example function for S_n with u as a leaf has depth two. The reason why we treat powers of $(1-z)$ in the denominator separately will shortly become clearer.

Now if ρ is a root of $I(G,z)$, then from Proposition A.1 we know that $f_u(\rho) = 1$. As mentioned in Section 1, to quantify the gap between $\beta(G)$ and the second smallest absolute value over the remaining roots of $I(G,z)$, involves two steps.

Our first result is to show using Proposition A.2 that $I(G,z)$ is injective in a neighborhood of $\beta(G)$. For this purpose, we define

$$r_G := \frac{\beta(G)^{\mathrm{dia}(G)}}{2n}, \tag{7}$$

where $\mathrm{dia}(G)$ is the diameter of G (see Appendix A). We show the following:

▶ **Theorem 2.1.** *The polynomial $I(G,z)$ is injective on $D(\beta(G), r_G/2)$, that is, $\beta(G)$ is the unique root of the polynomial in this disc.*

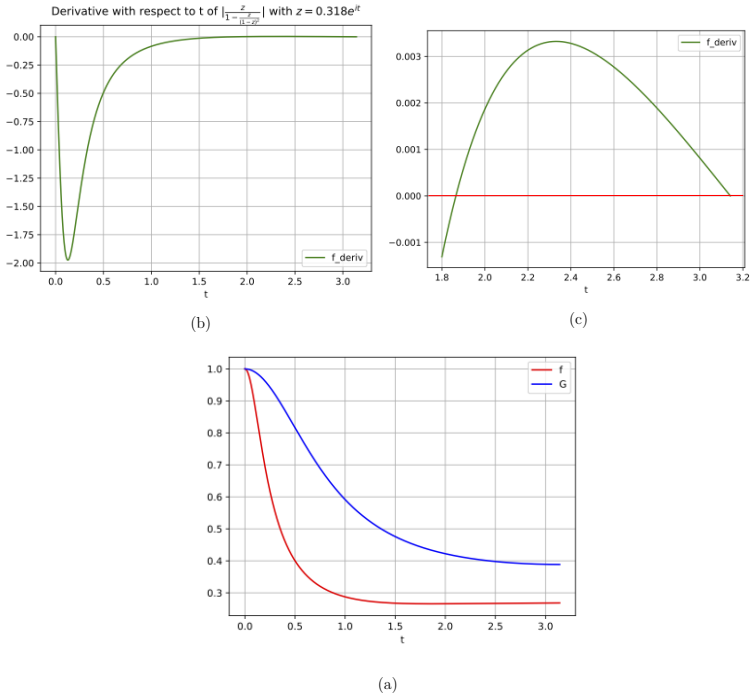


Figure 1 The absolute value function is not always monotone. In (a) we have the plot of $\left|z/(1-z/(1-z)^2)\right|$ in red color, corresponding to S_3 as given in Equation (4), where $z := \beta e^{it}$, and $\beta \sim 0.318$ is the point where the function takes the value one. The plot in blue color shows the corresponding majorizing function. (b) Shows the derivative with respect to t of the absolute value in $[0,\pi]$ and (c) shows the same graph zoomed in to highlight an additional root of the derivative besides 0 and π.

We next need to show that for the points on the circle $\beta(G)e^{i\theta}$ that are outside the disc $D(\beta(G), r_G/4)$, there is a disc centered around each of the points such that the value $|f_u(z)|$ is smaller than one on these discs. The following result makes this precise:

▶ **Lemma 2.2.** *If $|w - \beta(G)e^{i\theta}| \leq r_G(1 - \left|f_u(\beta(G)e^{i\theta})\right|)$ then $|f_u(w)| < 1$. In other words, there is no root of $I(G,z)$ in the disc $D(\beta(G)e^{i\theta}, r_G(1 - \left|f_u(\beta(G)e^{i\theta})\right|))$, for $\theta > 0$.*

Note that the radius of the disc goes to zero as θ approaches zero, but this case is already handled in Theorem 2.1. In order to make the bound explicit in terms of graph parameters, we need to upper bound $\left|f_u(\beta e^{i\theta})\right|$ for θ sufficiently far away from the origin. Ideally one would expect $\left|f_u(\beta e^{i\theta})\right|$ to monotonically decrease in θ as it varies from 0 to π, but this is not the case, as shown in Figure 1 for the function $z/(1-z/(1-z)^2)$.

Nevertheless, this is not far away from the truth, since as the depth of $f_u(z)$ increases it concentrates around the origin and drops sharply as θ increases; we are not able to prove this, but our observation is that it has the properties of a "good kernel" [11]. Instead, we show that there is a natural function that majorizes $|f_u(z)|$ on the boundary of the disc $D(0, re^{i\theta})$, within its domain of holomorphy, and that is also monotonically decreasing with θ.

A **majorant function** $G_r(\theta)$ for a complex valued function $g(re^{i\theta})$ satisfies the following two properties:

1. $G_r(0) = g(r)$.
2. For all θ, $|g(re^{i\theta})| \leq G_r(\theta)$, that is, the function majorizes g on the circle $re^{i\theta}$.
3. It is a monotone decreasing function, i.e., $G'_r(\theta) \leq 0$; attains its maximum at the origin, i.e., $G'_r(0) = 0$; and it is symmetric about the y-axis, i.e., $G_r(\theta) = G_r(-\theta)$.

For example, consider the function in the base case $z/(1-z)^\ell$. Its absolute value on $re^{i\theta}$ is $r/|1-re^{i\theta}|^\ell$, which by a simple calculation turns out to be

$$g_r(\theta) = \frac{r}{(1-2r\cos\theta+r^2)^{\ell/2}}.$$

In this case it is easier to argue monotonicity because the derivative with respect to θ is

$$\frac{-\ell r^2 \sin\theta}{(1-2r\cos\theta+r^2)^{(\ell+2)/2}},$$

which is negative for $\theta \in (0,\pi)$. However, we use a simpler majorant function that upper bounds the absolute value in the base case and behaves similarly. Based on the observation that $\left|1-re^{i\theta}\right| \geq 1-r\cos\theta$, one such function is

$$g_r(\theta) := \frac{r}{(1-r\cos\theta)^\ell}. \tag{8}$$

For $r \leq \beta(G-u)$, consider the function

$$f_u(re^{i\theta}) = \frac{re^{i\theta}}{(1-re^{i\theta})^\ell \prod_j (1-g_j(re^{i\theta}))},$$

where j varies over some fixed index, and $\ell \geq 0$. A majorant function $F_{u,r}(\theta)$ for $|f_u(re^{i\theta})|$ is obtained recursively from the majorant functions $G_{j,r}(\theta)$ for $\left|g_j(re^{i\theta})\right|$, respectively, as follows:

$$F_{u,r}(\theta) := \frac{r}{(1-r\cos\theta)^\ell \prod_j (1-G_{j,r}(\theta))}. \tag{9}$$

The reason we treat powers of $(1-re^{i\theta})$ separately is because if we take the absolute value inside, as we will immanently do for the g_j's, we will get a constant function r. So, in principle, we assume that the g_j's have *depth* more than one.

Let us verify that $F_{u,r}(\theta)$ satisfies all the properties of a majorant function. Firstly,

$$F_{u,r}(0) = \frac{r}{(1-r)^k \prod_j (1-G_{j,r}(0))}.$$

But as $G_{j,r}(0) = g_j(r)$ and the latter is positive it follows that

$$F_{u,r}(0) = \frac{r}{(1-r)^k \prod_j |1 - g_j(r)|} = f_u(r).$$

Secondly

$$|f_u(re^{i\theta})| \leq \frac{r}{|1 - re^{i\theta}|^k \prod_j (1 - |g_j(re^{i\theta})|)} \leq \frac{r}{(1 - r\cos\theta)^k \prod_j (1 - G_{j,r}(\theta))} = F_{u,r}(\theta)$$

and thirdly, taking the logarithmic derivative with respect to θ of $F_{u,r}(\theta)$ we obtain

$$\frac{F'_{u,r}(\theta)}{F_{u,r}(\theta)} = -\frac{kr\sin\theta}{(1 - r\cos\theta)} + \sum_j \frac{G'_{j,r}(\theta)}{(1 - G_{j,r}(\theta))}. \tag{10}$$

Therefore, the derivative is non-positive since by induction $G'_{j,r}(\theta) \leq 0$ and $F_{u,r}(\theta) > 0$; moreover, it also vanishes at $\theta = 0$, hence, the maximum value is attained at the origin which is equal to $f_u(r) \leq 1$, for $r \leq \beta(G)$; the symmetric nature also follows by applying induction to Equation (9).

The majorant function will be used in Lemma 2.2 instead of $\left|f_u(\beta e^{i\theta})\right|$. However, we still need a more explicit upper bound on $F_{u,r}(\theta)$. The monotone nature of the function comes to rescue, since locally around the origin we will show that the function is upper bounded by an inverted parabola, that is, $F_{u,r}(\theta) \leq F_{u,r}(0) - c\theta^2$ for $\theta \leq \theta_G$ and some constant c dependent on graph parameters. Therefore, the upper bound at θ_G holds for all $F_{u,r}(\theta)$, for $\theta > \theta_G$. Substituting this upper bound at θ_G in Lemma 2.2 then gives us the desired explicit disc around every point $\beta e^{i\theta}$ that is devoid of roots. In order to derive this local upper bound, we need to derive an upper bound on a variant of the gamma-function for $F_{u,r}(\theta)$. This is done inductively. Define $F_u(\theta) := F_{u,\beta(G)}(\theta)$, that is, $F_{u,r}(\theta)$ with $r = \beta(G)$, and $G_j(\theta) := G_{j,\beta(G)}(\theta)$. Then from Equation (9) it follows that

$$F_u(\theta) = \frac{\beta(G)}{(1 - \beta(G)\cos\theta)^\ell \prod_{j=1}^{d(u)-\ell}(1 - G_j(\theta))}. \tag{11}$$

In particular, we show the following result:

▶ **Lemma 2.3.** *For $F_u(\theta)$ defined as in Equation (11), and*

$$\Gamma := \max_{j=1}^{d} \sup_{m\geq 0} \left|\frac{G_j^{(m+1)}(0)}{(m+1)!}\right|^{1/(m+1)}$$

we have

$$\sup_{k\geq 0} \left|\frac{F_u^{(k)}(0)}{k!}\right|^{1/k} \leq \frac{2d\Gamma}{\beta(G)}.$$

As a consequence, we have

▶ **Lemma 2.4.** *If $\theta \leq (\beta(G)/2d)^{2\Delta}$, where d is the maximum degree of G, and Δ is the depth of $F_u(\theta)$, then*

$$F_u(\theta) \leq 1 - \frac{(\beta(G)\theta)^2}{4}.$$

To express the bound only in terms of d, we can use Shearer's bound $\lambda_S(d)$ instead of $\beta(G)$.

Substituting this in Lemma 2.2, we obtain

▶ **Corollary 2.5.** *Define*

$$\theta_G := \left(\frac{\beta(G)}{4n}\right)^{dia(G)}. \tag{12}$$

Then for all $\theta \geq \theta_G$ there is no root of $I(G,z)$ in the disc $D(\beta(G)e^{i\theta}, r_G(\beta(G)\theta)^2/4)$,

Finally, combining this result with Theorem 2.1 gives us Theorem 1.1 as desired.

In the next sections, we develop the proofs and details of the results above.

3 Univalence of $I(G,z)$ around $\beta(G)$

Throughout this section, we use $\beta := \beta(G)$, and $v \in V$ as a representative vertex. The function $f_v(z)$ given in Equation (3) can also be expressed as

$$f_v(z) = \frac{zI(G \setminus N[v], z)}{I(G \setminus v, z)} = 1 - \frac{I(G,z)}{I(G \setminus v, z)}. \tag{13}$$

Then $f_v(\beta) = 1$, $f_v(0) = 0$, and $f_v'(0) = 1$. The next result gives a lower bound on the growth of f_v in the vicinity of β_G.

▶ **Lemma 3.1.** *At the smallest root $\beta(G)$ of $I(G,z)$, we have $f_v'(\beta_G) > 1/\beta_G$.*

Proof. Taking the derivative on both sides of Equation (13), considering the second formulation, we get that

$$f_v'(z) = -\left(\frac{I(G,z)}{I(G \setminus v, z)}\right)' = -\frac{I'(G,z)}{I(G \setminus v, z)} + \frac{I(G,z)I'(G \setminus v, z)}{I(G \setminus v, z)^2}.$$

Since β is a root of $I(G)$, we get

$$f_v'(\beta) = -\frac{I'(G,\beta)}{I(G \setminus v, \beta)}.$$

Substituting Equation (29) for the derivative, we further obtain that

$$\begin{aligned} f_v'(\beta) &= \sum_{u \in V} \frac{I(G - N[u], \beta)}{I(G - v, \beta)} \\ &= \frac{I(G - N[v], \beta)}{I(G - v, \beta)} + \sum_{u \in V, u \neq v} \frac{I(G - N[u], \beta)}{I(G - v, \beta)} \\ &= \frac{1}{\beta} + \sum_{u \in V, u \neq v} \frac{I(G - N[u], \beta)}{I(G - v, \beta)}, \end{aligned} \tag{14}$$

where the last step follows from the definition of β. Now observe that the graphs $G \setminus N[u]$, for $u \in V \setminus \{v\}$ and $G \setminus v$ are all subgraphs of G. Therefore, from Proposition A.1 we know that their smallest root is larger than β and hence the corresponding independence polynomials evaluated at β are all positive. This means that the terms in the summation above are all positive, which gives us the desired lower bound on $f_v'(\beta)$. ◀

We also have a corresponding upper bound on $f_v'(\beta)$.

▶ **Lemma 3.2.** *For all $v \in V$, $f'_v(\beta) \leq \frac{n}{\beta^{dia(G)}}$, where $dia(G)$ is the diameter of G.*

Proof. Let the vertex set of the graph G be $V = v, v_1, \ldots, v_{n-1}$. Again consider Equation (14). We start with a lower bound on the denominator $I(G - v, \beta)$ as follows. We know that $I(G - v, \beta) = \beta I(G - N[v], \beta)$. Let $G_v = G - N[v] + v_1$ where $v_1 \in N[v]$, and β_v be the smallest root of $I(G_v, z)$. Now $I(G, z)$ is continuously decreasing on the real line starting from the origin down to its smallest root. At the origin we have $I'(G, 0) = |V(G)|$. Since G_v is a subgraph of G, we have $\beta_v > \beta$. Therefore,

$$I(G - v, \beta) = \beta I(G - N[v], \beta) = \beta I(G_v - v_1, \beta) \geq \beta I(G_v - v_1, \beta_v).$$

Repeating the above argument for vertex v_1 and so on we obtain $I(G - v, \beta) \geq \beta^k$, where k is the maximum distance of any vertex from the vertex v. Since the diameter $dia(G)$ of the graph is the longest shortest path in the graph and $\beta < 1$ we have

$$I(G - v, \beta) \geq \beta^{\mathrm{dia}(G)}. \tag{15}$$

Note that $I(G - N[u], \beta) < 1$ for all $u \in V$. Substituting these two bounds in Equation (14) we get

$$f'_v(\beta) \leq \frac{1}{\beta} + \frac{n-1}{\beta^{\mathrm{dia}(G)}} = \frac{\beta^{\mathrm{dia}(G)-1} + n - 1}{\beta^{\mathrm{dia}(G)}} \leq \frac{n}{\beta^{\mathrm{dia}(G)}},$$

since $\beta \leq 1$, $\beta^{\mathrm{dia}(G)} \leq 1$. ◀

We next derive an upper bound on higher-order derivatives of $I(G, z)$, which will be useful later.

▶ **Lemma 3.3.** *Let β be the smallest root of $I(G, z)$, then for all $0 \leq k \leq n$ we have $\left|I^{(k)}(G, \beta)\right| \leq \binom{n}{k} \leq n^k$. More generally, if H is a subgraph of G, then $I^{(k)}(H, \beta) \leq \binom{|H|}{k} \leq |H|^k$.*

Proof. For a general k, we have

$$I^{(k)}(G, z) = (-1)^k \sum_{\substack{u_1 \in V, u_2 \in G - N[u_1], \ldots, \\ u_k \in G - \cup_{j=1}^{k-1} N[u_j]}} I(G - N[u_1] - N[u_2] - \cdots N[u_k], z).$$

Since the graph $G - N[u_1] - N[u_2] - \cdots N[u_k]$ is a subgraph of G, its evaluation at β is smaller than one. The number of distinct choices of $u_1, \ldots, u_k$ are at most $\binom{n}{k}$, which completes the proof. ◀

Using the bounds above, we derive an upper bound on $\gamma_{I'(G,z)}$ (see Equation (32)), the standard gamma-function for the derivative $I'(G, z)$ at β. We start with deriving a lower bound on $I'(G, \beta)$: Since

$$I'(G, \beta) = -\sum_{u \in V} I(G - N[u], \beta)$$

and each $I(G - N[u], \beta)$ has the same sign and by Equation (15) is at least $\beta^{\mathrm{dia}(G)}$, we obtain

$$|I'(G, \beta)| \geq n\beta^{\mathrm{dia}(G)}. \tag{16}$$

It is not hard to see from the bound in Lemma 3.3 above and Equation (16) that

$$\gamma = \gamma_{I'(G,z)}(\beta) := \max_{k=1,\dots,n} \left| \frac{I^{(k+1)}(G,\beta)}{k!I'(G,\beta)} \right|^{1/k} \leq \max_{k=1,\dots,n} \left| \frac{n^{k+1}}{k!n\beta^{\mathrm{dia}(G)}} \right|^{1/k} \leq \frac{n}{\beta^{\mathrm{dia}(G)}}. \tag{17}$$

In order to apply Proposition A.2 to $I(G,z)$ centered at β, we first need to derive an upper bound on $|I'(G,z)|$ in a neighborhood of β. Consider the Taylor series expansion of the derivative around β

$$I'(G,z) = \sum_{k\geq 0} \frac{I^{(k+1)}(G,\beta)}{k!}(z-\beta)^k.$$

Taking absolute values and pulling out the constant term we get

$$|I'(G,z)| \leq |I'(G,\beta)| \sum_{k\geq 0} \left| \frac{I^{(k+1)}(G,\beta)}{k!I'(G,\beta)} \right| |z-\beta|^k.$$

Substituting the upper bound from Equation (17) in the right-side, we obtain that for $z \in D(\beta, r)$

$$|I'(G,z)| \leq |I'(G,\beta)| \sum_{k\geq 0} \gamma^k r^k = \frac{|I'(G,\beta)|}{(1-r\gamma)},$$

as long as $r\gamma < 1$. By Equation (7), we know that, $r(G)\gamma \leq 1/2$. Therefore, for all $z \in D(\beta, r_G)$, $|I'(G,z)| \leq 2\,|I'(G,\beta)|$. Substituting this upper bound, along with the definition of $r(G)$, in Proposition A.2 applied to $I(G,z)$ at β, we get Theorem 2.1.

4 Gap around every point on the circle with radius $\beta(G)$

In this section, we prove Lemma 2.2, that is, we show that the gap to unity for every $\theta > \theta_G$ is governed by the gap of $f_u(re^{i\theta})$ to $f_u(r)$ and a constant that depends on r. We will do the argument only for $r = \beta(G)$. For this purpose, we first need an upper bound on $\left|f_u^{(k)}(\beta(G))\right|$. Again, for convenience, let $\beta := \beta(G)$.

The kth derivative, up to sign, will have the form

$$\left(\frac{I(G,z)}{I(G-u,z)}\right)^{(k)} = \sum_{j=\max\{0,k-n\}}^{k} \binom{k}{j} I(G,z)^{(k-j)} \left(\frac{1}{I(G-u,z)}\right)^{(j)}. \tag{18}$$

Applying Arbogast's formula Equation (33) we obtain

$$\left(\frac{1}{I(G-u,z)}\right)^{(j)} = \sum_{i_1,\dots,i_j} \frac{(-1)^{i_1+\cdots+i_j}}{I(G-u,z)^{1+i_1+\cdots+i_j}} \frac{j!(i_1+\cdots+i_j)!}{i_1!\dots i_j!} \prod_{m=1}^{j} \left(\frac{I^{(m)}(G-u,z)}{m!}\right)^{i_m} \tag{19}$$

where the sum is over all indices $i_1, \dots, i_j$ such that

$$i_1 + 2i_2 + 3i_3 + \cdots + ji_j = j \tag{20}$$

Substituting $z = \beta$ in Equation (18), the term corresponding to $j = k$ disappears in the sum. Plugging the upper bound on the derivatives from Lemma 3.3 in Equation (19) above we get that

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \sum_{j=0}^{k-1} \frac{1}{j!(k-j)!} n^{(k-j)} \cdot \sum_{i_1,\dots,i_j} \frac{1}{I(G-u,\beta)^{1+i_1+\cdots+i_j}} \frac{j!(i_1+\cdots+i_j)!}{i_1!\dots i_j!} \prod_{m=1}^{j} \left(\frac{n^m}{m!}\right)^{i_m}.$$

From Equation (36), we obtain that $n^{\sum_m mi_m} = n^j$. Moreover, as $I(G-u,\beta) < 1$, we can upper bound its exponent by k as well to get

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \left(\frac{n}{I(G-u,\beta)}\right)^k \sum_{j=0}^{k-1} \frac{1}{j!(k-j)!} \sum_{i_1,\ldots,i_j} \frac{j!(i_1+\cdots+i_j)!}{i_1!\ldots i_j!} \prod_{m=1}^{j} \left(\frac{1}{m!}\right)^{i_m}.$$

Using Equation (35) the summation over the indices $i_1,\ldots,i_j$ can be expressed as

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \left(\frac{n}{I(G-u,\beta)}\right)^k \sum_{j=0}^{k-1} \frac{1}{j!(k-j)!} \sum_{t=0}^{j} t!B_{j,t}(1,\ldots,1),$$

The last summation over t is the ordered Bell number, $\widetilde{B}_j$ (see Appendix A), which gives us

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \left(\frac{n}{I(G-u,\beta)}\right)^k \sum_{j=0}^{k-1} \frac{\widetilde{B}_j}{j!(k-j)!}.$$

Furthermore, applying the upper bound from Equation (38) we derive that

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \left(\frac{n}{I(G-u,\beta)}\right)^k \sum_{j=0}^{k-1} \frac{2^j}{(k-j)!}.$$

Notice that the summation

$$\sum_{j=0}^{k-1} \frac{2^j}{(k-j)!} = 2^k \sum_{j=1}^{k} \frac{1}{j!2^j} < 2^k \sum_{j=1}^{\infty} \frac{1}{j!2^j} = 2^k(e^{1/2}-1) \leq 2^k.$$

Therefore, we finally obtain that

$$\frac{\left|f_u(\beta)^{(k)}\right|}{k!} \leq \left(\frac{2n}{I(G-u,\beta)}\right)^k.$$

As a consequence, we get that

$$\gamma_{f_u}(\beta) \leq \frac{2n}{I(G-u,\beta)} \leq \frac{2n}{\beta^{\mathrm{dia}(G)}} = \frac{1}{r_G}. \tag{21}$$

From this bound, we can derive the following estimate, an alternate proof of Theorem 2.1:

▶ **Lemma 4.1.** *For all $z \in D(0,\beta + r_G/2)$, $|f_u(z)| \leq 2$.*

Proof. A straightforward application of the triangle inequality to the Taylor series of f_u around β yields

$$|f_u(z)| \leq f_u(\beta) \sum_{k\geq 0} (r\gamma_{f_u}(\beta))^k = \frac{f_u(\beta)}{1 - r\gamma_{f_u}(\beta)}.$$

Since $r \leq r_G/2$ it follows that $r\gamma_{f_u}(\beta) \leq 2$, whence the upper bound on $|f_u(z)|$ in $D(\beta, r_G/2)$. Because of the maximum modulus principle, the upper bound holds on the whole disc. ◀

Let w be a point in the vicinity of $\beta(G)e^{i\theta}$. Then taking the Taylor expansion around $\beta(G)e^{i\theta}$, we get the following upper bound:

$$|f_u(w)| \leq \left|f_u(\beta e^{i\theta})\right| + \sum_{k\geq 1} \frac{|f_u^{(k)}(\beta e^{i\theta})|}{k!} |w - \beta e^{i\theta}|^k.$$

Since f_u is holomorphic with positive coefficients around the origin, from the strong maximum modulus principle, the maximum of $|f_u^{(k)}(\beta e^{i\theta})|$ for all θ is attained at the origin. Now, using the upper bound for $r = \beta$ from Equation (21), we obtain that

$$|f_u(w)| < \left|f_u(\beta e^{i\theta})\right| + \sum_{k\geq 1}\left(\frac{|w-\beta e^{i\theta}|}{r_G}\right)^k.$$

Define $r := |w - \beta e^{i\theta}|/r_G$. If $r < 1$ then using the formula for a geometric series we have

$$|f_u(w)| < \left|f_u(\beta e^{i\theta})\right| + \frac{r}{1-r}.$$

Therefore, as long as

$$\frac{r}{1-r} \leq 1 - \left|f_u(\beta e^{i\theta})\right|$$

we have $|f_u(w)| < 1$, or equivalently, if

$$|w - \beta e^{i\theta}| \leq r_G\left(\frac{1-\left|f_u(\beta e^{i\theta})\right|}{2-|f_u(\beta e^{i\theta})|}\right)$$

the function cannot take the value one in the vicinity of $\beta e^{i\theta}$. The denominator can be simplified to one since the maximum value of $\left|f_u(\beta e^{i\theta})\right|$ is one at the origin. This completes the proof of the following Lemma 2.2.

5 Upper bound on the Majorant Function near the origin

In this section, we give the proofs of Lemma 2.3 and Lemma 2.4. We again use the shorthand $\beta := \beta(G)$. The idea is to consider the Taylor series expansion of $F_{u,r}(\theta)$ around the origin. More precisely, we have

$$F_{u,r}(\theta) = F_{u,r}(0) + F'_{u,r}(0)\theta + \sum_{k\geq 2}\frac{F^{(k)}_{u,r}(0)}{k!}\theta^k.$$

Since the first derivative vanishes, we have

$$F_{u,r}(\theta) = F_{u,r}(0) + \sum_{k\geq 2}\frac{F^{(k)}_{u,r}(0)}{k!}\theta^k.$$

In fact, all the odd derivatives vanish and, the second derivative is negative. We first verify the latter condition. From Equation (10) it follows that

$$F^{(2)}_{u,r}(\theta) = F'_{u,r}(\theta)\sum_j \frac{G'_{j,r}(\theta)}{(1-G_{j,r}(\theta))} + F_{u,r}(\theta)\sum_j\left(\frac{G^{(2)}_{j,r}(\theta)}{(1-G_{j,r}(\theta))} + \frac{(G'_{j,r}(\theta))^2}{(1-G_{j,r}(\theta))^2}\right). \quad (22)$$

Now inductively, the second derivatives $G^{(2)}_{j,r}(0)$ are negative (the base case from Equation (8) is $-r^2/(1-r)^2$), all the other terms vanish, which yield us that $F^{(2)}_{u,r}(\theta)(0) < 0$ as desired. This means that locally near the origin $F_{u,r}(\theta) \sim F_{u,r}(0) - c\theta^2$. We will next show that for θ sufficiently small, half of the second term will dominate the remaining summation in absolute value, and so $F_{u,r}(\theta) \leq F_{u,r}(0) - F^{(2)}_{u,r}(0)\theta^2/4$ for $\theta \leq \theta_G$. For this purpose we need an upper bound on the absolute values of the kth derivatives of $F_{u,r}$ at the origin with $r = \beta$, which will be used to derive an upper bound on the γ-function for $F_{u,r}$. The upper bound will be derived inductively.

Let us begin with recalling some definitions: From Equation (11) we have

$$F_u(\theta) = \frac{\beta}{\prod_{j=1}^{d(u)}(1 - G_j(\theta))},$$

where we have simplified the denominator to subsume the functions $\beta \cos\theta$ in the product by appropriate indexing, and $G_j(\theta) := G_{j,\beta}(\theta)$. We next derive a formula for the kth derivative of $F_u(\theta)$.

It can be verified that for $k \geq 1$,

$$F_u^{(k)}(\theta) = \sum_{\ell=0}^{k-1} \binom{k-1}{\ell} F_u^{(k-1-\ell)}(\theta) \sum_j -(\ln(1 - G_j(\theta)))^{(\ell+1)}. \tag{23}$$

Using Arbogast's formula, Equation (33), for the functions $\ln(1 - G_j(\theta))$, along with Equation (41), we further get that

$$F_u^{(k)}(\theta) = \sum_{\ell=0}^{k-1} \binom{k-1}{\ell} F_u^{(k-1-\ell)}(\theta) \sum_j \sum_{i_0,\ldots,i_\ell} \frac{(\ell+1)!}{i_0!\ldots i_\ell!} \frac{(i_0 + \cdots + i_\ell)!}{(1 - G_j(\theta))^{\sum_{m=0}^{\ell} i_m}} \prod_{m=0}^{\ell} \left(\frac{G_j^{(m+1)}(\theta)}{(m+1)!} \right)^{i_m},$$

where $i_0, \ldots, i_\ell$ is an $(\ell+1)$-tuple of non-negative integers satisfying the equation

$$i_0 + 2i_1 + \cdots + (\ell+1)i_\ell = \ell + 1. \tag{24}$$

At this stage, we can inductively argue that if k is odd then the derivative at the origin vanishes; this is because one of the indices m will be odd and by induction $G_j^{(m+1)}(0)$ vanishes. Dividing both sides by $k!$, simplifying the binomial $\binom{k-1}{\ell}$ term, and substituting $\theta = 0$ we obtain that

$$\frac{F_u^{(k)}(0)}{k!} = \sum_{\ell=0}^{k-1} \frac{F_u^{(k-1-\ell)}(0)}{k(k-1-\ell)!\ell!} \sum_j \sum_{i_0,\ldots,i_\ell} \frac{(\ell+1)!}{i_0!\ldots i_\ell!} \frac{(i_0 + \cdots + i_\ell)!}{(1 - G_j(0))^{\sum_{m=0}^{\ell} i_m}} \prod_{m=0}^{\ell} \left(\frac{G_j^{(m+1)}(0)}{(m+1)!} \right)^{i_m}, \tag{25}$$

Define

$$\Gamma := \max_{j=1}^{d} \sup_{m \geq 0} \left(\frac{\left| G_j^{(m+1)}(0) \right|}{(m+1)!} \right)^{1/(m+1)}, \tag{26}$$

which implies that

$$\frac{\left| G_j^{(m+1)}(0) \right|}{(m+1)!} \leq \Gamma^{m+1}.$$

Furthermore, we inductively assume that

$$\max_{j=0}^{k-1} \left(\frac{\left| F_u^{(j)}(0) \right|}{j!} \right)^{1/j} \leq \frac{2d\Gamma}{\beta}.$$

Since $F_u(0) = 1$, we also know that $\prod_j (1 - G_j(0)) = \beta$, which implies that for all j, $(1 - G_j(0)) \geq \beta$. Taking the absolute value, applying the triangle inequality, and substituting these upper and lower bounds in the right-hand side, we get the following

$$\frac{\left| F_u^{(k)}(0) \right|}{k!} \leq \sum_{\ell=0}^{k-1} \left(\frac{2d\Gamma}{\beta} \right)^{(k-1-\ell)} \frac{1}{k\ell!} \sum_j \sum_{i_0,\ldots,i_\ell} \frac{(i_0 + \cdots + i_\ell)!(\ell+1)!}{i_0!\ldots i_\ell!} \prod_{m=0}^{\ell} \left(\frac{\Gamma^{(m+1)}}{\beta} \right)^{i_m}$$

From Equation (24), the term $\prod_m \Gamma^{i_m(m+1)} = \Gamma^{\ell+1}$ and hence

$$\frac{\left|F_u^{(k)}(0)\right|}{k!} \leq \Gamma^k \sum_{\ell=0}^{k-1} \left(\frac{2d}{\beta}\right)^{(k-1-\ell)} \frac{(\ell+1)!}{k\ell!} \sum_j \sum_{i_0,\ldots,i_\ell} \frac{(i_0+\cdots+i_\ell)!}{i_0!\ldots i_\ell!} \left(\frac{1}{\beta}\right)^{\sum_m i_m}$$
$$= \Gamma^k \sum_{\ell=0}^{k-1} \left(\frac{2d}{\beta}\right)^{(k-1-\ell)} \frac{(\ell+1)}{k} \sum_j \sum_{i_0,\ldots,i_\ell} \frac{(i_0+\cdots+i_\ell)!}{i_0!\ldots i_\ell!} \prod_{m=0}^{\ell} \left(\frac{1}{\beta}\right)^{\sum_m i_m}.$$

The summation term over j is independent of it, so we can upper bound the summation by the degree d. Furthermore, if we define $t = i_0 + \cdots + i_\ell$, for a fixed t, then the right-hand side further simplifies to

$$\frac{\left|F_u^{(k)}(0)\right|}{k!} \leq d\Gamma^k \sum_{\ell=0}^{k-1} \left(\frac{2d}{\beta}\right)^{(k-1-\ell)} \frac{(\ell+1)}{k} \sum_{t=1}^{\ell+1} \frac{1}{\beta^t} \sum_{\substack{i_0,\ldots,i_\ell \\ i_0+\cdots+i_\ell=t}} \frac{t!}{i_0!\ldots i_\ell!}$$
$$\leq d\Gamma^k \sum_{\ell=0}^{k-1} \left(\frac{2d}{\beta}\right)^{(k-1-\ell)} \frac{(\ell+1)}{k} \sum_{t=1}^{\ell+1} \frac{1}{\beta^t} \binom{\ell+1}{t-1},$$

where the last step follows from Equation (39). Since $\beta \leq 1$, we can use $\beta^{-(\ell+1)}$ instead of β^{-t} to get

$$\frac{\left|F_u^{(k)}(0)\right|}{k!} \leq d\left(\frac{\Gamma}{\beta}\right)^k \sum_{\ell=0}^{k-1} (2d)^{(k-1-\ell)} \frac{(\ell+1)}{k} \sum_{t=1}^{\ell+1} \binom{\ell+1}{t-1}$$
$$\leq d\left(\frac{2\Gamma}{\beta}\right)^k \sum_{\ell=0}^{k-1} \frac{(\ell+1)}{k} d^{(k-1-\ell)},$$

where in the last step we upper bound the summation of the binomials by $2^{\ell+1}$. Since $d \geq 2$, it can be showed that the new summation term is at most d^{k-1}, which finally yields us the desired claim in Lemma 2.3.

In the base case Γ is the standard gamma-function for the cosine function, which is smaller than $1/\sqrt{2} < 1$. Therefore, we get

$$\frac{\left|F_u^{(k)}(0)\right|}{k!}^{1/k} \leq \left(\frac{2d}{\beta}\right)^{\Delta},$$

where Δ is the depth of $f_u(z)$.

In order for half of the second term in the Taylor series expansion of $F_u(\theta)$ around the origin to dominate the sum of the remaining terms, we want

$$\left|\frac{F_u^{(2)}(0)}{4}\right| \theta^2 \geq \sum_{\ell \geq 2} \left(\frac{2d}{\beta}\right)^{2\Delta\ell} \theta^{2\ell}.$$

Assuming

$$\theta^2 \leq (\beta/2d)^{\Delta}/2, \tag{27}$$

the above inequality follows if

$$\theta^2 \leq \frac{\left|F_u^{(2)}(0)\right|}{4} \left(\frac{\beta}{2d}\right)^{\Delta}. \tag{28}$$

We next derive an explicit lower bound on the second-derivative.

Recall that the first derivatives $F'_u(0)$ and $G'_j(0)$ vanish, and that the second derivatives $G^{(2)}_j(0)$ are all negative. Therefore, from Equation (22) we obtain that

$$F^{(2)}_u(0) = \sum_j \frac{G^{(2)}_j(0)}{(1-G_j(0))} \geq \sum_j G^{(2)}_j(0).$$

Therefore, $\left|F^{(2)}_u(0)\right| \geq d \min \left|G^{(2)}_j(0)\right|$. Inductively, we obtain that

$$\left|F^{(2)}_u(0)\right| \geq d^{\Delta-1}\beta^2/(1-\beta)^{d+1},$$

since the absolute value of the second derivative of the majorant function in the base case is $d\beta^2/(1-\beta)^{d+1}$. Substituting this in Equation (28), we get a slightly weaker constraint than Equation (27), namely, $\theta^2 \leq \beta^{\Delta+2}/4d$. So, in order to simplify, we combine the two constraints to obtain Lemma 2.4.

In order to combine this lemma with Theorem 2.1, we notice that the disc $D(\beta, r_G/2)$ subtends the angle $\arcsin(r_G/2\beta)$, which is at least $\beta^{\mathrm{dia}(G)}/(4n)$, at the origin. Take θ_G as a quantity smaller than this bound and the constraint in Lemma 2.4, namely as defined in Equation (12). Since the function $F_u(\theta)$ is monotonically decreasing, we know that for all $\theta \geq \theta_G$, $F_u(\theta) \leq F_u(\theta_G)$. Substituting this in Lemma 2.2, we obtain that for all $\theta \geq \theta_G$, the disc

$$D(\beta e^{i\theta}, r_G(\beta\theta_G)^2/4)$$

is devoid of roots. Since the radius here is smaller than $r_G/2$, we combine this with Theorem 2.1 to finally obtain that the disc

$$D\left(0, \beta + \left(\frac{\beta}{n}\right)^{O(\mathrm{dia}(G))}\right)$$

contains exactly one root of $I(G,z)$ completing the proof of Theorem 1.1. Note that the depth Δ is smaller than the diameter $\mathrm{dia}(G)$, which instead is bounded by n.

6 Some Explicit Lower Bounds On The Gap

In this section, we derive explicit lower bounds on the gap between the smallest root $\beta(G)$ and the root with the second smallest absolute value of the independence polynomial of some fundamental graph classes. In particular, we give the explicit lower bounds for the Path Graph (P_n), the Cycle Graph (C_n) on n vertices and the Complete Bipartite Graph ($K_{n\times n}$) on $2n$ vertices. For convenience, let $\beta := \beta(G)$ and $\alpha := \alpha(G)$ be the root with the second smallest absolute value of the independence polynomial of graph G in each of the cases.

Path Graph

To describe the independence polynomial $P_n(z)$ of the Path Graph we need the Fibonacci polynomials [7]: Let $F_1(z) := 1$, $F_2(z) := z$ and recursively define $F_{n+1}(z) := zF_n(z) + F_{n-1}(z)$. From the relation Equation (2) for $P_n(z)$ we obtain that $z^{n-1}P_n(-1/z^2) = F_{n+2}(z)$. The roots of $F_{n+2}(z)$ are $2i\cos(k\pi/(n+2))$, $k = 1, \ldots, n+1$. Therefore, the roots of $P_n(z)$ are $1/(4\cos^2(k\pi/(n+2)))$, for $k = 1, \ldots, \lfloor \frac{n+1}{2} \rfloor$. Therefore, $\beta = \frac{1}{4\cos^2\left(\frac{\pi}{m}\right)}$ and $\alpha = \frac{1}{4\cos^2\left(\frac{2\pi}{m}\right)}$, with $m := n+2$. Using the Taylor series for the cosine function, for large values of m, we get

$$\frac{\cos\left(\frac{\pi}{m}\right)}{\cos\left(\frac{2\pi}{m}\right)} = 1 + \frac{3\pi^2}{2m^2} + O(m^{-4}).$$

On squaring, we obtain that

$$\frac{\alpha}{\beta} = \frac{\cos^2\left(\frac{\pi}{m}\right)}{\cos^2\left(\frac{2\pi}{m}\right)} = 1 + \frac{3\pi^2}{m^2} + O(m^{-4}) = 1 + \Omega\left(\frac{1}{n^2}\right).$$

Cycle Graph

The independence polynomial $C_n(z)$ of the Cycle Graph can be expressed in terms of the Chebyshev polynomial of the first kind $T_n(z)$ [9]:

$$C_n(z) = 2z^{n/2}T_n\left(\frac{1}{2\sqrt{z}}\right).$$

Therefore, its roots are $\frac{1}{4\cos^2\left(\frac{(2k+1)\pi}{2n}\right)}$, $k = 0, 1, \ldots, \lfloor\frac{n-1}{2}\rfloor$. This implies that $\beta = \frac{1}{4\cos^2\left(\frac{\pi}{2n}\right)}$ and $\alpha = \frac{1}{4\cos^2\left(\frac{3\pi}{2n}\right)}$. An argument similar to the one above implies that asymptotically we have

$$\frac{\alpha}{\beta} = 1 + \frac{2\pi^2}{n^2} + \frac{17\pi^4}{6n^4} + O(n^{-6}) = 1 + \Omega\left(\frac{1}{n^2}\right).$$

Complete Bipartite Graph

The independence polynomial $K_{n\times n}(z)$ of the Complete Bipartite Graph on $2n$ vertices is $K_{n\times n}(z) = 2(1-z)^n - 1$. Its roots are $z_k = 1 - 2^{-\frac{1}{n}}e^{\frac{i2k\pi}{n}}$, $k = 0, 1, \ldots, n-1$. Therefore $\beta = 1 - 2^{-\frac{1}{n}}$ and $\alpha = 1 - re^{i\frac{2\pi}{n}}$, where $r := 2^{-\frac{1}{n}}$. Since, in this case α is a complex root, we compute ratio of the absolute values of the roots:

$$\frac{|\alpha|}{\beta} = \frac{\sqrt{1 + r^2 - 2r\cos(\frac{2\pi}{n})}}{1-r} = \frac{\sqrt{(1-r)^2 + 4r\sin^2(\theta/2)}}{1-r} = \sqrt{1 + \frac{4r}{(1-r^2)}\sin^2(\theta/2)}.$$

Take $b := \ln(2)$ then, $r = e^{-b/n} = 1 - \frac{b}{n} + \frac{b^2}{2n^2} + O(n^{-3})$ and $\sin(\frac{\pi}{n}) = \frac{\pi}{n} - \frac{\pi^3}{6n^3} + O(n^{-5})$. After substituting these estimates and simplifying we get that for large n

$$\begin{aligned}\frac{|\alpha|}{\beta} &= \sqrt{1 + \frac{4\pi^2}{b^2} + \left(\frac{-\pi^2}{3} - \frac{4\pi^4}{3b^2}\right)\frac{1}{n^2} + O(n^{-3})} \\ &= \sqrt{1 + \frac{4\pi^2}{b^2}} - \frac{\pi^2}{6}\frac{1}{n^2} + O(n^{-3}) \\ &\approx 9.119 - O\left(\frac{1}{n^2}\right).\end{aligned}$$

7 Concluding Remarks

This paper provides the first quantitative lower bound on the gap between the smallest root of the independence polynomial and the second smallest absolute value. A simple proof for the existence of the gap can be based on the maximum modulus principle. Quantifying the principle for the special function at hand involves studying its local behaviour. To the best of our knowledge, this is the first time a result is provided for separation between roots of

the independence polynomial; earlier results always focus on zero-free regions. Our larger hope is to study the algorithmic implications of the ratio of $\beta(G)$ to the second smallest absolute value. Can it be used to design algorithms that are efficient for those graphs where this ratio is large, for example, the graph classes mentioned in Section 6?

References

1 Ferenc Bencs, Péter Csikvári, Piyush Srivastava, and Jan Vondrák. On complex roots of the independence polynomial. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 675–699, 2023. doi:10.1137/1.9781611977554.ch29.

2 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer-Verlag, New York, 1998.

3 Péter Csikvári. Note on the smallest root of the independence polynomial. *Combinatorics, Probability and Computing*, 22:1–8, 2012. doi:10.1017/S0963548312000302.

4 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, USA, 1 edition, 2009.

5 Massimiliano Goldwurm and Massimo Santini. Clique polynomials have a unique root of smallest modulus. *Information Processing Letters*, 75(3):127–132, 2000. doi:10.1016/S0020-0190(00)00086-7.

6 Lawrence A. Harris. On the size of balls covered by analytic transformations. *Monatshefte für Mathematik*, 83(1):9–23, March 1977. doi:10.1007/BF01303008.

7 V. E. Hoggatt Jr. and Marjorie Bicknell. Roots of Fibonacci Polynomials. *The Fibonacci Quarterly*, 11(3):271–274, 1973. doi:10.1080/00150517.1973.12430825.

8 Steven G. Krantz and Harold R. Parks. *A Primer of Real Analytic Functions*. Birkhäuser, 2012. doi:10.1007/978-0-8176-8134-0.

9 Vadim E Levit and Eugen Mandrescu. The independence polynomial of a graph–a survey. In *Proceedings of the 1st International Conference on Algebraic Informatics*, volume 233254, pages 231–252. Aristotle Univ. Thessaloniki Thessaloniki, 2005.

10 J.B. Shearer. On a problem of Spencer. *Combinatorica*, 5:241–245, 1985. doi:10.1007/BF02579368.

11 Elias M. Stein and Rami Shakarchi. *Fourier Analysis: An Introduction*. Princeton University Press, 2003.

A Basic Results

Throughout this paper $G = (V, E)$ will be assumed to be a simple undirected graph. Let n be the number of vertices in G. For every vertex $u \in V$, let $d(u)$ denote its degree, and $d := \max_u d(u)$, for all u; we will assume that $d \geq 2$. Let $\mathrm{dia}(G)$ denote the **diameter of** G, that is the length of the longest path between any pair of vertices in any connected component of G. In the subsequent sections, we will need some basic properties of the independence polynomial (see [3, 1] for proofs):

▶ **Proposition A.1.** *Let G be as above and $I(G, z)$ be its independence polynomial.*

1. *The derivative $I'(G, z)$ satisfies*

$$I'(G, z) = -\sum_{u \in V} I(G - N[u], z). \tag{29}$$

2. *If H is a subgraph of G then $\beta(G) \leq \beta(H)$.*

3. *Shearer [10] showed a lower bound on $\beta(G)$, namely,*

$$\beta(G) \geq \lambda_S(d) := \frac{(d-1)^{d-1}}{d^d}. \tag{30}$$

Applying the third property with G as the complete graph and H an arbitrary graph with at most n vertices, we also have the following lower bound

$$\beta(H) \geq \frac{1}{n}. \tag{31}$$

We will also need variants of Smale's gamma-function [2]: Given a function $f : \mathbb{C} \to \mathbb{C}$ holomorphic at a point $z \in \mathbb{C}$, such that $f^{(j)}(z) \neq 0$, define

$$\gamma_{f,j}(z) := \sup_{k>j} \left| \frac{j!f^{(k)}(z)}{k!f^{(j)}(z)} \right|^{1/(k-j)}. \tag{32}$$

The standard gamma-function corresponds to $\gamma_{f,0}(z)$, and we will simply use $\gamma_f(z)$ to denote that. Intuitively, the function is related to the inverse of the radius of convergence of $f^{(j)}(z)$ at z.

We recall Arbogast's formula (also called the formula of Faà di Bruno) [8] for derivatives of composition of functions: For $N \in \mathbb{Z}_{\geq 0}$, the Nth derivative of

$$(f \circ g)^{(N)}(z) = \sum_{i_1,\ldots,i_N} \frac{N!}{i_1! \ldots i_N!} f^{(i_1+\cdots+i_N)}(g(z)) \prod_{m=1}^{N} \left(\frac{g^{(m)}(z)}{m!} \right)^{i_m}, \tag{33}$$

where the sum is over all tuples of non-negative integers $i_1, \ldots, i_N$ such that

$$i_1 + 2i_2 + 3i_3 + \cdots + Ni_N = N.$$

Given a K, we can combine the terms corresponding to $i_1 + \cdots + i_N = K$ and simplify the summation as follows. Since $\sum_j j i_j = N$ it follows with the additional constraint that $\sum_j (j-1) i_j = N - K$. This implies that $i_j = 0$, for $j > N - K + 1$, and so we can express Equation (33) as

$$(f \circ g)^{(N)}(z) = \sum_{K=0}^{N} \sum_{i_1,\ldots,i_N} f^{(K)}(g(z)) B_{N,K}(g'(z), g^{(2)}(z), \ldots, g^{(N-K+1)}(z)), \tag{34}$$

where $B_{N,K}(x_1, \ldots, x_{N-K+1})$ are **the partial exponential Bell polynomials**:

$$B_{N,K}(x_1, \ldots, x_{N-K+1}) := \sum_{i_1,\ldots,i_{N-K+1}} \frac{N!}{i_1! \ldots i_{N-K+1}!} \prod_{m=1}^{N-K+1} \left(\frac{x_m}{m!} \right)^{i_m}, \tag{35}$$

and $i_1, \ldots, i_{N-K+1}$ satisfy the following two constraints:

$$\begin{aligned} i_1 + 2i_2 + 3i_3 + \cdots + (N-K+1)i_{N-K+1} &= N \text{ and} \\ i_1 + i_2 + i_3 + \cdots + i_{N-K+1} &= K. \end{aligned} \tag{36}$$

Notice that $B_{N,K}(1, \ldots, 1) = \left\{ {N \atop K} \right\}$, the Sterling number of second kind, that is number of ways to partition an N element set into K non-empty parts, and hence

$$\sum_{K=0}^{N} K! B_{N,K}(1, \ldots, 1) = \widetilde{B}_N$$

the **ordered Bell number**, which satisfy the following recurrence:

$$\widetilde{B}_N = \sum_{i=1}^{N-1} \binom{N}{i} \widetilde{B}_i. \tag{37}$$

From this, we can derive the following claim inductively:

$$\frac{\widetilde{B}_N}{N!} \leq \left(\frac{1}{\ln 2}\right)^N. \tag{38}$$

We will also need the following observation:

$$\sum_{i_1,\ldots,i_{N-K+1}} \frac{K!}{i_1!\ldots i_{N-K+1}!} = \binom{N-1}{K-1} \tag{39}$$

where the sum is over all tuples $(i_1, \ldots, i_N)$ satisfying the conditions in Equation (36). The left-hand side counts all partition of N into K blocks where the ordering of distinct blocks only matter (distinct blocks correspond to different choices of j). However, these are precisely the number of compositions of N into K parts, which is the term on the right-hand side.

Besides the above, we need the following observations for $j \geq 0$

$$\left(\frac{1}{x}\right)^{(j)} = \frac{(-1)^j j!}{x^{j+1}} \tag{40}$$

and

$$(\ln x)^{(j)} = \left(\frac{1}{x}\right)^{(j-1)} = \frac{(-1)^{j-1}(j-1)!}{x^j}. \tag{41}$$

Given $\alpha \in \mathbb{C}$ and $r > 0$, we will denote by $D(\alpha, r) \subset \mathbb{C}$ the open disc centered at α with radius r.

In addition to the above, we need the following quantified result on the injectiveness of a function in a neighborhood of a point where the derivative does not vanish [6]:

▶ **Proposition A.2.** *Let $h : D(\alpha, r) \to \mathbb{C}$ be a holomorphic function such that $h(\alpha) = 0$ and $h'(\alpha) \neq 0$. Then h is injective on the disc*

$$D\left(\alpha, \frac{r|h'(\alpha)|}{\sup_{z \in D(\alpha,r)} |h'(z)|}\right).$$

On the Hardness of Order Finding and Equivalence Testing for ROABPs

C. Ramya
The Institute of Mathematical Sciences, Chennai, India

Pratik Shastri
The Institute of Mathematical Sciences, Chennai, India

Abstract

The complexity of representing a polynomial by a Read-Once Oblivious Algebraic Branching Program (ROABP) is highly dependent on the chosen variable ordering. Bhargava et al. [5] prove that finding the optimal ordering is NP-hard, and provide some evidence (based on the Small Set Expansion hypothesis) that it is also hard to approximate the optimal ROABP width. In another work, Baraskar et al. [3] show that it is NP-hard to test whether a polynomial is in the GL_n orbit of a polynomial of sparsity at most s.

Building upon these works, we show the following results: first, we prove that approximating the minimum ROABP width up to any constant factor is NP-hard, when the input is presented as a circuit. This removes the reliance on stronger conjectures in the previous work [5]. Second, we show that testing if an input polynomial given in the sparse representation is in the affine GL_n orbit of a width-w ROABP is NP-hard. Furthermore, we show that over fields of characteristic 0, the problem is NP-hard even when the input polynomial is homogeneous. This provides the first NP-hardness results for membership testing for a dense subclass of polynomial sized algebraic branching programs (VBP). Finally, we locate the source of hardness for the order finding problem at the lowest possible non-trivial degree, proving that the problem is NP-hard even for quadratic forms.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases ROABP, Order Finding, Equivalence Testing, NP-hardness, Hardness of Approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.49

Funding *C. Ramya*: Research supported by INSPIRE Faculty Fellowship of DST.

1 Introduction

Algebraic circuits provide a powerful framework for understanding the complexity of computing multivariate polynomial over a field. These are directed acyclic graphs whose in-degree 0 vertices are labeled by variables or field constants and internal vertices are either addition or multiplication gates. Several computational problems concerning polynomials such as identity testing, polynomial factoring, equivalence testing and reconstruction have been studied intensively for various structured subclasses of circuits.

In this article, we are interested in *Read-Once Oblivious Algebraic Branching Programs* (ROABPs for short), the algebraic analogs of ordered binary decision diagrams (OBDDs). Informally, an ROABP is layered DAG with a designated source and sink vertex whose edges are labeled by univariate polynomials. More importantly, all edges in the same layer use the same variable and a variable occurs in exactly one layer. The polynomial computed by the ROABP is the sum over all source-to-sink paths of the product of edge weights along each path. In this work, we study the computational complexity of two problems concerning ROABPs: *order-finding* and *equivalence testing*. Before delving into the details of these two problems, we note that in algorithms concerning polynomials, a polynomial $f(x_1, \ldots, x_n)$ of degree d can be given as an input to an algorithm in one of the following standard representations:


© C. Ramya and Pratik Shastri;
licensed under Creative Commons License CC-BY 4.0

- Dense Representation: a list (of size $\binom{n+d}{d}$) of coefficients for all possible monomials up to degree d.
- Sparse Representation: a list of pairs, where each pair consists of a non-zero coefficient and its corresponding monomial (represented by its exponent vector).
- Circuit Representation: an algebraic circuit computing f.
- Black-Box Access: an oracle access to evaluations of f.

It is important to note that the choice of representation can critically affect the complexity of a computational problem. The input representations above are ordered in increasing order of compactness. As a rule, problems considered in this paper become harder when the input is presented more compactly. Conversely, proving hardness becomes easier.

First, we begin with the order finding problem for ROABPs. It is easy to see that with every ROABP computing a polynomial $f(x_1, \ldots, x_n)$ we can associate a unique permutation $\sigma : [n] \to [n]$ which we call the *order* of the ROABP. The size of an ROABP, particularly its *width*, which is the maximum number of vertices in any of its layers serves as a key measure of complexity. A crucial feature of ROABPs is that the width required to compute a polynomial f is critically dependent on the order. A poor choice of ordering can lead to an exponential blow-up in the required width compared to an optimal one. For instance, the minimal widths of ROABPs computing $(x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n)$ in orders $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$ and $(x_1, \ldots, x_n, y_1, \ldots, y_n)$ are drastically different. This sensitivity gives rise to the natural computational question of order finding introduced in Bhargava et al. [5]:

▶ **Problem 1** (CktROWidth-d). *[Decision version of order-finding]: Given a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ of degree at most d as an algebraic circuit and an integer w in binary, decide whether there exists an ROABP for f of width at most w, in some order.*

▶ **Problem 2** (SearchCktROWidth-d). *[Search version of order-finding]: Given an algebraic circuit C computing an n-variate polynomial f of degree $\leq d$, find an order $\sigma \in S_n$ that minimizes the ROABP width for f.*

Depending on the input representation, other variations of these problems are defined analogously. For instance, DenseROWidth-d is the decision version of the problem when the input is provided in the dense representation.

Bhargava, Dutta, Ghosh, and Tengse [5] study the complexity of the order-finding problem. In particular, they show that DenseROWidth-6 is NP-hard. Their proof is an interesting reduction from the *cutwidth* problem for graphs. This is a *linear arrangement problem* in which given a graph the goal is to find an ordering (a.k.a linear arrangement) of the vertices that minimizes the maximum number of edges between any prefix and the corresponding suffix in that ordering. It is known that the cutwidth problem is NP hard even for graphs with maximum degree 3. Their reduction is parameter preserving: The graph is transformed into a polynomial in such a way that ROABP width of the polynomial is 2 more than the cutwidth of the graph. The degree of the polynomial constructed is exactly 2Δ, where Δ denotes the maximum degree of the graph.

Furthermore, Bhargava et al. [5] also study the problem of approximating ROABP width. They provide two pieces of evidence that indicate hardness of approximation, both stemming from their reduction. First, as noted earlier, the reduction in [5] is parameter preserving. Hence, hardness of approximation for the cutwidth problem directly translates to hardness of approximation for ROABP width. Austrin et al. [2] showed that cutwidth is hard to approximate assuming the *small set expansion* (SSE) hypothesis thus implying hardness of approximating ROABP width assuming the SSE hypothesis, even when the input is given in the dense representation. Second, they develop a *tensoring technique* through which

they show that a constant factor approximation for Search-CktROWidth would imply a PTAS for Search-CktROWidth, and therefore, by their reduction, for cutwidth. This is (potentially stronger) evidence against the existence of a constant factor approximation for Search-CktROWidth, since Ambühl et al. [1] show that for a problem related to cutwidth, a PTAS does not exist under assumptions weaker than SSE (but stronger than $\mathsf{P} \neq \mathsf{NP}$).

In this work, we prove NP-Hardness of approximating Search-CktROWidth up to any arbitrary constant factor.

▶ **Theorem 3** (Inapproximability of Search-CktROWidth-d)**.** *Let $\alpha \in \mathbb{R}$ be an arbitrary constant. Let $n, d \in \mathbb{N}$ be given as input, in unary. Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ of degree d be given as input, as an arithmetic circuit. Let $w \in \mathbb{N}$ also be give as input, in binary. It is* NP*-hard to α-approximate the width of the smallest ROABP for f.*

Our reduction is natural, direct, and again, from cutwidth. It works over any field. We use the power of the circuit representation to construct, from a graph G, a depth three circuit for a polynomial f_G such that for every subset S of vertices, rank of the Nisan matrix of f_G with respect to the set S is exactly $2^{|\mathrm{cut}(S)|}$. For a subset of vertices S, $\mathrm{cut}(S)$ denotes the number of edges going out of S. See Section 2 for definition of Nisan matrices. This already gives NP-hardness of 2-approximation. Then, we use the tensoring technique from [5] to get hardness of α-approximation for any constant α.

Next, we move on to the problem of testing equivalence to ROABPs. Polynomial equivalence testing is a well-studied problem in algebraic complexity theory: given two polynomials the goal is to decide if one is equivalent to the other via an invertible affine transformation of the variables. Several special cases of the polynomial equivalence problem have been studied time and again. In order to understand recent progress on this problem, we consider the notion of *orbits* of polynomial families. Let $\mathbf{x}$ denote $(x_1, \ldots, x_n)$. The *orbit* of an n-variate polynomial f is the set of polynomials obtained from f by applying an invertible affine transformation to the variables, i.e., $\mathsf{orbit}(f) = \{f(A\mathbf{x} + \mathbf{b}) \mid A \in \mathrm{GL}_n(\mathbb{F}) \text{ and } \mathbf{b} \in \mathbb{F}^n\}$. For any class C of polynomials, the orbit of C is the union of orbits of polynomials in C. The orbits of the determinant and permanent polynomials are central to geometric complexity theory.

In the equivalence problem for a certain class C of circuits we are given a polynomial f (in some representation) and the goal is to determine if f is in the (GL_n) orbit of a certain circuit class C. In other words, decide if f affine equivalent (under invertible transformations) to some polynomial in C. Medini and Shpilka [9] study the orbit of the continuant polynomial which is the trace of a certain product of matrices of dimension two and design polynomial time reconstruction algorithms for the same.

In the context of testing whether a given polynomial promised to be in the orbit of a certain circuit class is identically zero or not, Medini and Shpilka [9] construct hitting sets for orbits of read-once formulas[1] and certain *dense* subclasses of depth three circuits. Saha and Thankey [12] designed hitting sets for orbits of ROABPs. Recently, Bhargava and Ghosh [6] obtained smaller hitting sets for the same class of polynomials. It is known [12] that orbits of polynomial size ROABPs are a dense subclass of the the class of polynomial size *general* algebraic branching programs.

Gupta et al. [8] consider the equivalence problem for read-once arithmetic formulas and give a randomized polynomial-time algorithm (with oracle access to quadratic form equivalence) for the same over fields of characteristic zero. Baraskar et al. [4] give a randomized

[1] Read-once formulas are arithmetic formulas where every variable appears as a leaf at most once.

algorithm to test equivalence to *design polynomials*[2] over fields of sufficiently large size and characteristic. In a recent, beautiful work, Baraskar et al. [3] show NP hardness of testing equivalence to *sparse polynomials* over any field when the input polynomial is given in the sparse representation. The problem is as follows: given f in the sparse representation and in integer w, does there exist an $A \in \mathrm{GL}_n$, $\mathbf{b} \in \mathbb{F}^n$ and a polynomial $g(\mathbf{x})$ with at most w non-zero monomials such that $f = g(A\mathbf{x} + \mathbf{b})$? They also show that a related problem, that of deciding equivalence to constant support polynomials, is NP hard.

In this work, we consider the problem of testing equivalence to ROABPs.

▶ **Problem 4** (ROABP-Equivalence). *Given an n-variate polynomial $f \in \mathbb{F}[x_1 \ldots, x_n]$ in its sparse representation and an integer w in binary, decide if there exists an $A \in GL_n(\mathbb{F})$ and a $\mathbf{b} \in \mathbb{F}^n$ such that $f(A\mathbf{x} + \mathbf{b})$ has an ROABP of width at most w.*

We show that this problem is NP-hard over all fields. Over fields of characteristic 0, it remains hard even when the input polynomial is homogeneous. To the best of our knowledge, this provides the first NP-hardness result for membership testing for a dense subclass of polynomial size ABPs:

▶ **Theorem 5.** *Over fields of characteristic 0, the* ROABP-Equivalence *problem is* NP*-hard even when the input polynomial f is homogeneous. Over fields of prime characteristic, the* ROABP-Equivalence *problem is* NP*-hard.*

Our proof combines ideas from the papers of Baraskar et al. [3] and Bhargava et al. [5]. Specifically, we reduce from cutwidth and construct a polynomial f such that the linear transformation which minimizes the ROABP width of $f(A\mathbf{x} + \mathbf{b})$ is always the product of a diagonal matrix and a permutation matrix.

In our final result, we pinpoint the hardness of the original order-finding problem to the simplest non-trivial class of polynomials. We show that ROABP order finding is NP-hard even when the input is restricted to be a quadratic form (Theorem 25).

▶ **Theorem 6.** *The problem DenseROWidth-2 is NP-hard over all fields.*

Previously, NP-hardness was known for polynomials of degree ≥ 6 [5]. To show this, we reduce from a different, more algebraic linear arrangement problem called linear rank-width (Problem 14).

2 Preliminaries

We now formally define the concepts central to our results, including ROABPs, Nisan's characterization, and the graph theoretic computational problems we reduce from.

2.1 ROABPs and Width Characterization

▶ **Definition 7** (Read-Once Oblivious ABP (ROABP)). *Let $\mathbb{F}$ be a field. An ROABP R computing an n-variate polynomial $f(x_1, \ldots, x_n)$ over $\mathbb{F}$ in a variable order $\sigma \in S_n$ is a layered, directed graph with $n + 1$ layers, indexed 0 to n.*

- *The 0^{th} layer contains a single source vertex s, and the n^{th} layer contains a single sink vertex t.*

[2] Design polynomials are a special class of polynomials in which the degree of the GCD of every pair of monomials is bounded.

- *Edges only exist between adjacent layers, from layer $i-1$ to layer i for $i \in [n]$.*
- *Edges between layer $i-1$ and i are labeled with univariate polynomials in the variable $x_{\sigma(i)}$.*

The polynomial computed by the ROABP is the sum of products of edge weights over all paths from s to t.

▶ **Definition 8** (Width of an ROABP). *The* width *of an ROABP is the maximum number of vertices in any of its layers. For a polynomial f and an order σ, the ROABP-width,* $\text{RO-width}_\sigma(f)$*, is the width of the minimal-width ROABP for f in order σ. Then* $\text{RO-width}(f)$ *is defined as* $\min_{\sigma \in S_n} \text{RO-width}_\sigma(f)$.

Nisan's work [10] provides an exact algebraic characterization of ROABP width, as the rank of a certain matrix of coefficients.

▶ **Definition 9** (Nisan Matrix). *For a polynomial $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ and a set of variables $X_T = \{x_i\}_{i \in T}$, the* Nisan Matrix *$M_T(f)$ is a matrix whose rows are indexed by monomials in variables from X_T and columns by monomials in variables from $\{x_1, \ldots, x_n\} \setminus X_T$. The (m_1, m_2)-th entry is the coefficient of $m_1 \cdot m_2$ in f.*

▶ **Theorem 10** (Nisan's Characterization [10]). *For any polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ and order σ, the number of vertices in the ith layer of an optimal ROABP for f in order σ is exactly* $\text{rank}(M_{T_i}(f))$*, where $T_i = \{\sigma(1), \ldots, \sigma(i)\}$.*

Next, we define two graph layout problems. They will be central to our reductions.

2.2 Graph Layout Problems

The authors of [5] prove that DenseROwidth-d is NP hard (for $d \geq 6$) via a reduction from a particular NP-hard graph layout problem called Cutwidth. We define this problem next.

▶ **Definition 11** (Cutwidth [7]). *Given a graph $G = ([n], E)$, a* linear arrangement *is a permutation $\pi : [n] \to [n]$. The* cutwidth *of G with respect to π is* $\max_{i \in [n-1]} |\text{cut}(\{\pi(1), \ldots, \pi(i)\})|$[3]*. The* Cutwidth *of G is the minimum cutwidth over all arrangements.*

It is known [7] that the following problem is NP hard, even for graphs with maximum degree ≤ 3:

▶ **Problem 12** (CutWidth [7]). *Given a graph $G = ([n], E)$ with maximum degree ≤ 3 and an integer $w \in \mathbb{N}$ in binary, decide whether the cutwidfth of G is at most w.*

Next we define a similar looking graph layout problem, called Linear Rank-Width. Here, the cut size is replaced by the rank of a certain matrix.

▶ **Definition 13** (Linear Rank-Width$_\mathbb{F}$ [11]). *Let $G = ([n], E)$ be a graph and let $\pi : [n] \to [n]$ be a linear arrangement. For $i \in [n-1]$, let A_i be a matrix over $\mathbb{F}$ with rows indexed by vertices $\{v \mid \pi(v) \leq i\}$ and columns by vertices $\{v \mid \pi(v) > i\}$. The entry (u, v) is 1 if $\{u, v\} \in E$ and 0 otherwise. The* linear rank-width *of G with respect to π is* $\max_{i \in [n-1]} \text{rank}(A_i)$*. The linear rank-width of G is the minimum linear rank-width over all arrangements.*

It is known [11] that the the analogous decision problem of minimizing linear rank-width is also NP-hard, over any field.

▶ **Problem 14** (Linear Rank-Width$_\mathbb{F}$ [7]). *Given a graph $G = ([n], E)$ and an integer $w \in \mathbb{N}$ in binary, decide whether the linear rank-width of G over $\mathbb{F}$ is at most w.*

[3] For a graph $G = ([n], E)$ and a subset $S \subseteq [n]$ of vertices, $\text{cut}(S)$ denotes the number of edges with one endpoint in S and the other in $[n] \setminus S$

3 Inapproximability of ROABP Order-Finding

In this section, we show the NP-hardness of approximating ROABP width up to an arbitrary constant factor, when the input is a circuit. We will require the following Lemma of Bhargava et al.

▶ **Lemma 15** ([5]). *Given a polynomial $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ with individual degree d and an $l \in \mathbb{N}$, define*

$$f^{\otimes l}(x_1, \ldots, x_n) = \prod_{k=0}^{l-1} f\left(x_1^{(d+1)^k}, \ldots, x_n^{(d+1)^k}\right)$$

For every subset $S \subseteq [n]$, we have $\mathrm{Rank}(M_S(f^{\otimes l})) = \mathrm{Rank}(M_S(f))^l$.

▶ **Theorem 16** (Inapproximability of Search-CktROwidth). *Let $\alpha \in \mathbb{R}$ be an arbitrary constant. Let $n, d \in \mathbb{N}$ be given as input, in unary. Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ of degree d be given as input, as an arithmetic circuit. Let $w \in \mathbb{N}$ also be give as input, in binary. It is NP-hard to distinguish between the following two cases:*

1. *f has an ROABP of width $\leq w$ in some order.*
2. *Every ROABP for f has width $> \alpha \cdot w$.*

Proof. We reduce from CutWidth for graphs with maximum degree ≤ 3. Given a graph $G = ([n], E)$, we first construct a small $\Pi\Sigma\Pi$ circuit computing a polynomial $f_G(x_1, \ldots, x_n)$ such that for any $S \subseteq [n]$, $\mathrm{Rank}(M_S(f_G)) = 2^{|\mathrm{cut}(S)|}$. For a vertex $i \in [n]$ and a neighbour j of i, let $n_i(j)$ denote the number of neighbours of i less than or equal to j.

Define

$$f_G(x_1, \ldots, x_n) = \prod_{\{i,j\} \in E} \left(1 + x_i^{n_i(j)} x_j^{n_j(i)}\right) = \sum_{T \subseteq E} \left(\prod_{\{i,j\} \in T} x_i^{n_i(j)} x_j^{n_j(i)} \right)$$

Let $S \subseteq [n]$ be arbitrary. Define the following sets of edges: $E_1 = \{\{i,j\} \mid \{i,j\} \in E \text{ and } i,j \in S\}$, $E_2 = \{\{i,j\} \mid \{i,j\} \in E \text{ and } i,j \in [n] \setminus S\}$ and $E_3 = \mathrm{cut}(S) = E \setminus (E_1 \cup E_2)$. Then, the non-zero rows of $M_S(f_G)$ are indexed by monomials m of the following type: m is characterized by a subset $E_1' \subseteq E_1$ and a subset $E_3' \subseteq E_3$ such that $m = \left(\prod_{\{i,j\} \in E_1'} x_i^{n_i(j)} x_j^{n_j(i)} \right) \left(\prod_{\substack{\{i,j\} \in E_3' \\ i \in S}} x_i^{n_i(j)} \right)$. Call E_3' the subset of cut edges *picked* by such a monomial/row. Similarly, the non-zero columns of $M_S(f_G)$ are indexed by monomials m' which are characterized by a subset $E_2' \subseteq E_2$ and a subset $E_3' \subseteq E_3$ such that $m' = \left(\prod_{\{i,j\} \in E_2'} x_i^{n_i(j)} x_j^{n_j(i)} \right) \left(\prod_{\substack{\{i,j\} \in E_3' \\ j \in [n] \setminus S}} x_j^{n_j(i)} \right)$

Observe the following:

- For a row indexed by m and column indexed by m', if the subset of cut edges picked by m and m' are not identical, $M_S(f_G)(m, m') = 0$.
- The submatrix induced by row and column monomials that pick the same subset of cut edges has rank 1, since every row in this matrix is all 1's.

Therefore $M_S(f_G)$ is a block diagonal matrix with $2^{|\mathrm{cut}(S)|}$ disjoint rank 1 blocks and so $\mathrm{Rank}(M_S(f_G)) = 2^{|\mathrm{cut}(S)|}$. Next, set $l = \lceil \log \alpha \rceil + 1$ and consider the polynomial $f_G^{\otimes l}$. It has degree $\leq (6|E|+1)^{\lceil \log \alpha \rceil + 1} - 1$, it has a formula of size $O_\alpha(|E|)$ which can be computed from G in polynomial time, and by Lemma 15, for every $S \subseteq [n]$ it satisfies $\mathrm{Rank}(M_S(f_G^{\otimes l})) = 2^{l \cdot |\mathrm{cut}(S)|}$. Therefore, it holds that

- If G has cutwidth $\leq k$, then RO-width$(f_G^{\otimes l}) \leq 2^{l \cdot k}$
- If G has cutwidth $\geq k+1$, then RO-width$(f_G^{\otimes l}) \geq 2^l \cdot 2^{l \cdot k} > \alpha 2^{l \cdot k}$.

This finishes the proof of Theorem 16. ◀

4 Hardness of Equivalence Testing for ROABPs

Bhargava et al. [5] demonstrate NP-hardness of order-finding problem for ROABPs via a reduction from the cutwidth problem for graphs. More precisely, given a graph $G = ([n], E)$ with maximum degree Δ, the authors of [5] construct the polynomial $f_G \in \mathbb{F}[x_1, \ldots, x_n]$ defined as

$$f_G = \sum_{\{i,j\} \in E} x_i^{n_i(j)} x_j^{n_j(i)} + \sum_{i=1}^{n} x_i^{\Delta+1} \tag{1}$$

where $n_i(j) \in [\Delta]$ is the number of neighbours of i less than or equal to j. The claim ([5], Claim 4.4) central to their reduction is that for every $S \subseteq [n]$, $\mathrm{Rank}(M_S(f_G)) = |\mathrm{cut}(S)| + 2$. This implies that the ROABP width of the polynomial f_G is exactly two more the cutwidth of the graph G and the reduction is *order preserving*, i.e., an optimal arrangement of the vertices in G is exactly an optimal order for an ROABP computing f_G.

In this section, we prove that over all fields, testing equivalence to width w ROABPs is NP-hard. Over fields of characteristic 0, we show that this problem is NP-hard even when the input polynomial is *homogeneous*, whereas in positive characteristic, we require inhomogeneity. Our reductions build on the reductions in Bhargava et al. [5] and Baraskar et al. [3]. In particular, given a graph $G = ([n], E)$, we construct a polynomial $f_G \in \mathbb{F}[x_1, \ldots, x_n]$ with the following two properties:

1. If $A \in \mathrm{GL}_n(\mathbb{F})$ is a permutation matrix times a diagonal matrix, then for every $\mathbf{b} \in \mathbb{F}^n$, the ROABP width of $f(A\mathbf{x} + \mathbf{b})$ is cutwidth$(G) + 2$.
2. If $A \in \mathrm{GL}_n(\mathbb{F})$ is not a permutation matrix times a diagonal matrix, then for every $\mathbf{b} \in \mathbb{F}^n$, the ROABP width of $f(A\mathbf{x} + \mathbf{b})$ is at least $|E| + 3$ in *every* order.

Property 2 essentially forces the width minimizing A to have a nice form, namely it is a permutation matrix times a diagonal matrix. This is because cutwidth$(G) \leq |E|$. With this overall plan, we proceed with the details of the reduction.

4.1 Characteristic Zero

Over characteristic 0, we construct a *homogeneous* f_G. In order to show property 2 for a homogeneous f_G, we need the following Lemma from [3] that provides a lower bound on the sparsity of a polynomial divisible by a high power of a linear form with support at least 2.

▶ **Lemma 17** ([3]). *Let $\mathbb{F}$ be a field of characteristic 0. Let $l \in \mathbb{F}[x_1, \ldots, x_n]$ be a linear polynomial with support ≥ 2, let $h \in \mathbb{F}[x_1, \ldots, x_n]$ be arbitrary and let $d \in \mathbb{N}$. Then $l^d \times h$ has at least $d+1$ non-zero monomials.*

In fact, we need a strengthening of this lemma. In the sequel, we strengthen this lemma and show that any polynomial divisible by the dth power of a linear form with support at least 2 has *ROABP width* at least $d+1$, in every order.

▶ **Lemma 18.** *Let $\mathbb{F}$ be a field of characteristic zero. Let ℓ be a linear polynomial with support at least 2 and $h \in \mathbb{F}[x_1, \ldots, x_n]$ be any non-zero polynomial and $d \in \mathbb{N}$. Let $F = \ell^d \cdot h$. Then for any $\sigma \in S_n$,* RO-width$_\sigma(f) \geq d+1$.

Proof. Without loss of generality, let x_1, x_2 be in the support of ℓ, i.e., $\ell = a_1x_1 + a_2x_2 + \ell'$ such that $a_1, a_2 \neq 0$. Then, $F = \ell^d \cdot h = (a_1x_1 + a_2x_2 + \ell')^d \cdot h$. We can view h as a polynomial in $\mathbb{F}(x_3, \ldots, x_n)[x_1, x_2]$, i.e., a polynomial in x_1, x_2 with coefficients in the field $\mathbb{F}(x_3, \ldots, x_n)$. Let k be the degree of h (over $\mathbb{F}(x_3, \ldots, x_n)$). That is, k is the maximal $i+j$ such that h contains the monomial $c_{i,j}x_1^i x_2^j$ with $c_{i,j} \in \mathbb{F}(x_3, \ldots, x_n)$ and $c_{i,j} \neq 0$. We denote by h_k the homogeneous component of h of degree k. Then, $F = F_1 + F_2$ where $F_1 = (a_1x_1 + a_2x_2)^d \cdot h_k$. Every monomial m in F_1 satisfies $\deg_{x_1}(m) + \deg_{x_2}(m) = d + k$. Furthermore, every monomial in F_2 satisfies $\deg_{x_1}(m) + \deg_{x_2}(m) < k + d$. Next, by applying Lemma 17 to $(a_1x_1 + a_2x_2)^d \times h_k$ *over the field* $\mathbb{F}(x_3, \ldots, x_n)$, we get $(a_1x_1 + a_2x_2)^d \times h_k = \sum_{\substack{i,j \\ i+j=d+k}} c_{i,j}x_1^i x_2^j$ such that $c_{i,j} \in \mathbb{F}[x_3, \ldots, x_n]$ and at least $d+1$ of the $c_{i,j}$'s non-zero. Let $m_{i,j}$ be the leading monomial of $c_{i,j}$. Observe that:

- There exists a set P of at least $d+1$ pairs (i,j) with $i+j = d+k$ such that for each pair (i,j) in P, the coefficient of the monomial $x_1^i x_2^j m_{i,j}$ in F_1 is non-zero.
- There is no monomial m in F with non-zero coefficient such that $\deg_{x_1}(m) + \deg_{x_2}(m) > d + k$.

Now, consider any ROABP for F, in an arbitrary order $x_{\pi(1)}, \ldots, x_{\pi(n)}$. Let S be a *prefix* of π that separates 1 and 2. Without loss of generality, assume that $1 \in S$ and $2 \in [n] \setminus S$. Since width of the ROABP is at least Rank$(M_S)(F)$ we now prove that Rank$(M_S(F)) \geq d+1$. For a monomial $m \in \mathbb{F}[x_1, \ldots, x_n]$, let $m[S]$ denote the monomial obtained from m by setting variables outside $\{x_i \mid i \in S\}$ to 1. Consider the submatrix of $M_S(F)$ induced by the row monomials $\{m_{i,j}[S]x_1^i \mid (i,j) \in P\}$ (order the rows by increasing values of i) and the column monomials $\{m_{i,j}[[n] \setminus S]x_2^j \mid (i,j) \in P\}$ (order the columns by increasing value of j). By our observations above, this is a full rank, square, anti-triangular matrix with non-zero entries on the main anti-diagonal. It has at least $d+1$ rows. Therefore, the rank of this submatrix is at least $|P| \geq d+1$ and so is the rank of $M_S(F)$. ◀

▶ **Theorem 19.** *The Equivalence to ROABP problem (Problem 4) is* NP*-hard over fields of characteristic* 0*.* NP*-hardness holds even when the input polynomial f is homogeneous.*

Proof. We reduce from CutWidth. Let $(G = ([n], E), w)$ be an instance of CutWidth. We map it to an instance $(f_G, w+2)$ of ROABP-Equivalence. To this end, introduce a total order $e_1 < e_2 < \ldots < e_{|E|}$ on the edges of G. For $i \in E$, let $e_i = \{i_1, i_2\}$ such that $i_1 < i_2$. Let $\mathbf{x} = (x_1, \ldots, x_n)$. For $S \subseteq [n]$, define $\Pi_S(\mathbf{x}) \triangleq \left(\prod_{j \in S} x_j^{|E|+2}\right)$ and define the polynomial

$$f_G(\mathbf{x}) \triangleq \Pi_{[n]}(\mathbf{x}) \left(\sum_{i=1}^{|E|} x_{i_1}^i x_{i_2}^{2|E|-i+1} \right)$$

Clearly, f_G is a homogeneous polynomial of degree $(n+2)|E| + 2n + 1$. We now prove both the forward and reverse directions of the reduction.

Forward Direction: If G has cutwidth at most w, then there exist $A \in \mathrm{GL}_n(\mathbb{F})$ and $\mathrm{b} \in \mathbb{F}^n$ such that $f_G(A\mathbf{x} + \mathrm{b})$ has an ROABP of width at most $w+2$ in some order. The proof of the forward direction follows the outline of the corresponding proof in [5]. In particular, we pick $A = I_n$, the $n \times n$ identity matrix, and $\mathbf{b} = 0$. Consider any subset $S \subseteq [n]$ and f_G ca be expressed as

$$f_G(\mathbf{x}) = \underbrace{\left(\Pi_S(\mathbf{x}) \sum_{\substack{i\in[|E|]\\ i_1,i_2\in S}} x_{i_1}^i x_{i_2}^{2|E|-i+1}\right)\Pi_{[n]\setminus S}(\mathbf{x})}_{f_1}$$
$$+ \underbrace{\left(\Pi_{[n]\setminus S}(\mathbf{x}) \sum_{\substack{i\in[|E|]\\ i_1,i_2\in[n]\setminus S}} x_{i_1}^i x_{i_2}^{2|E|-i+1}\right)\Pi_S(\mathbf{x})}_{f_2}$$
$$+ \underbrace{\left(\sum_{\substack{i\in[|E|]\\ \{i_1,i_2\}\in \mathrm{cut}(S)}} \Pi_S(\mathbf{x})\Pi_{[n]\setminus S}(\mathbf{x}) x_{i_1}^i x_{i_2}^{2|E|-i+1}\right)}_{f_3}$$

Observe that f_1 and f_2 are non-zero polynomials of the form $g(\mathbf{x}_S) \times h(\mathbf{x}_{[n]\setminus S})$, where for a subset S of $[n]$, $\mathbf{x}_S$ are the x variables indexed by S. Therefore, we have that $\mathrm{Rank}(M_S(f_1)) = \mathrm{Rank}(M_S(f_2)) = 1$. Finally, notice that f_3 can be written as a *sum* of $|\mathrm{cut}(S)|$ monomials. For every monomial m, $\mathrm{Rank}(M_S(m)) = 1$ for each S. Since $M_S(f_G) = M_S(f_1) + M_S(f_2) + M_S(f_3)$, by subadditivity of rank, we have that $\mathrm{Rank}(M_S(f_G)) \leq |\mathrm{cut}(S)| + 2$. Consider a linear arrangement $\pi : [n] \to [n]$ that witnesses $\mathrm{cutwidth}(G) \leq w$. Due to the above reasoning, combined with Nisan's characterization, we have that f_G has an ROABP in order π, of width $\leq w + 2$.

Reverse Direction: If there exist $A \in \mathrm{GL}_n(\mathbb{F})$ and $\mathrm{b} \in \mathbb{F}^n$ such that $f_G(A\mathrm{x} + \mathrm{b})$ has an ROABP of width at most $w + 2$ in some order, then G has cutwidth at most w.
In order to prove the reverse direction, we need the following key lemma.

▶ **Lemma 20.** *Suppose $A \in \mathrm{GL}_n(\mathbb{F})$ is not the product of a permutation matrix and a diagonal matrix, then for* every $\mathbf{b} \in \mathbb{F}^n$*, every ROABP computing $f_G(A\mathbf{x} + \mathbf{b})$ must have width at least $|E| + 3$.*

Proof. If A is not the product of a permutation matrix and a diagonal matrix, then $A\mathbf{x} + \mathbf{b}$ must send at least one x variable to a linear polynomial with support *at least* 2. We may then write $f_G(A\mathbf{x} + \mathbf{b}) = l(\mathbf{x})^{|E|+2} \times h$ for a nonzero polynomial h. By Lemma 18, for any $\sigma \in S_n$, $\text{RO-width}_\sigma(f_G(A\mathbf{x} + \mathbf{b})) \geq |E| + 3$. ◀

We use Lemma 20 to complete the reverse direction of the reduction. First, we observe that if A is the product of a diagonal matrix and a permutation matrix, then $\text{RO-width}_\sigma(f_G(A\mathbf{x}+b)) = \text{RO-width}_\sigma(f_G(\mathbf{x}))$: Suppose A has this form. Then there exist $a_1, \ldots, a_n \in \mathbb{F}$, all non-zero, and a permutation $\pi : [n] \to [n]$ such that $A\mathbf{x}+\mathbf{b} = (a_1x_{\pi(1)}+b_1, a_2x_{\pi(2)}+b_2, \ldots, a_nx_{\pi(n)}+b_n)$. We can obtain an ROABP for $f_G(\mathbf{x})$ from an ROABP for $f_G(a_1x_{\pi(1)} + b_1, \ldots, a_nx_{\pi(n)} + b_n)$ by replacing each $x_{\pi(i)}$ with $(x_i - b_i)/a_i$. The resulting ABP is still an ROABP, with the same width as before. Also, this process is clearly reversible.

By the proof of the forward direction, we know that $\text{RO-width}(f_G) \leq \mathrm{cutwidth}(G) + 2 \leq |E| + 2$. Now suppose there exist $A \in \mathrm{GL}_n$ and $\mathbf{b} \in \mathbb{F}^n$ such that $f_G(A\mathbf{x} + \mathbf{b})$ has ROABP width at most $w + 2$ in some order. Due to Lemma 20 and the observation above, we may

assume that A is the identity matrix and $\mathbf{b} = \mathbf{0}$. Next, we show that for each $S \subseteq [n]$, $\text{Rank}(M_S(f_G)) \geq |\text{cut}(S)| + 2$ by a proof similar to the NP hardness in [5] by exhibiting a submatrix of $M_S(f_G)$ that is a $|\text{cut}(S)| \times |\text{cut}(S)|$ permutation matrix, along with two rows that lie in disjoint spaces.

This suffices for the reverse direction, for if f_G has an ROABP of width $\leq w+2$ in order π, then due to Nisan's characterization (Theorem 10), we would have that $\text{cutwidth}(G) \leq w$, witnessed by the linear arrangement π.

Define $E_1 \triangleq \{e_i \mid e_i \in \text{cut}(S), i_1 \in S, i_2 \in [n] \setminus S\}$ and $E_2 \triangleq \{e_i \mid e_i \in \text{cut}(S), i_1 \in [n] \setminus S, i_2 \in S\}$. We look at the submatrix of $M_S(f_G)$ induced by the row monomials $R = \{\Pi_S(\mathbf{x}) x_{i_1}^i \mid e_i \in E_1\} \cup \{\Pi_S(\mathbf{x}) x_{i_2}^{2|E|+1-i} \mid e_i \in E_2\}$ and column monomials $C = \{\Pi_{[n]\setminus S}(\mathbf{x}) x_{i_1}^i \mid e_i \in E_2\} \cup \{\Pi_{[n]\setminus S}(\mathbf{x}) x_{i_2}^{2|E|+1} \mid e_i \in E_1\}$. This is a permutation matrix, since each monomoial labeling both the rows an columns can be associated with a unique end point of an edge in $\text{cut}(S)$, and the only non-zero entry (in the entirety of $M_S(f_G)$, not just the submatrix) in that row/column corresponds to the monomial labeling the column/row associated with the other end point of that edge. On the other hand, consider the row labeled by $\Pi_S(\mathbf{x})$. This has non-zero entries in the columns labeled by the monomials $\Pi_{[n]\setminus S}(\mathbf{x}) x_i^{2|E|+1}$, for each $i \in [n] \setminus S$ (note that these monomials are *not* contained in C), and a zero entry in the column labeled by $x_{[n]\setminus S}$. Also, a row labeled by $\Pi_S(\mathbf{x}) x_i^{2|E|+1}$ for an $i \in [S]$ (again, note that this monomial is *not* contained in R) has a non-zero entry in the column labeled by $\Pi_{[n]\setminus S}$. This gives us that $\text{Rank}(M_S(f_G)) \geq |\text{cut}(S)| + 2$. ◀

4.2 Characteristic p

In this section, we prove hardness of testing equivalence to width w ROABPs, over characteristic p. In this setting, we resort to inhomogeneity to prove hardness. This is because Lemma 17, and therefore, Lemma 20 fail to hold over small characteristic. In particular, we show that a skewed version of the polynomial in Equation (1) gives us hardness, even over characteristic p. Recall the polynomial $f_G = \sum_{\{i,j\}\in E} x_i^{n_i(j)} x_j^{n_j(i)} + \sum_{i=1}^n x_i^{\Delta+1}$ constructed in [5]. Instead of the term $\sum_{i=1}^n x_i^{\Delta+1}$, we introduce the asymmetric $\sum_{i=1}^n x_i^{D_j}$. We carefully choose distinct exponents D_j such that they are polynomial in the size of the graph while also allowing us to prove NP-hardness.

We will use the following well known result of Lucas.

▶ **Theorem 21.** *Let p be a prime and m, n be integers such that $m = \sum_{k=0}^t m_k p^k$ and $n = \sum_{k=0}^t n_k p^k$ are the base p expansions of m and n respectively. Then,*

$$\binom{m}{n} \equiv \prod_{k=0}^{t} \binom{m_k}{n_k} \mod p$$

where we use the convention that $\binom{m_k}{n_k} = 0$ if $m_k < n_k$.

▶ **Theorem 22.** *Let p be a prime and let $\mathbb{F}$ be a field of characteristic p. The Equivalence to ROABP problem (Problem 4) is* NP*-hard over $\mathbb{F}$.*

Proof. Let $(G = ([n], E), w)$ be an instance of the CutWidth problem with maximum degree of G is ≤ 3. We map it to an instance $(f_G, w+2)$ of ROABP-Equivalence.

Let $M = \max\{|E| + 4, 7\}$. Let L be the smallest integer such that $p^L > M$. For each $j \in [n]$, define the exponent

$$D_j := (p^L - 1) + (j-1)p^L.$$

Note that each D_j is $\mathsf{poly}(n, |E|)$. For each vertex i, let $n_i(j) \in [\deg(i)]$, as before, be number of neighbours of i less than or equal to j. Note that $n_i(j) \leq 3$ for each $\{i, j\} \in E$. Let $\mathbf{x} = (x_1, \ldots, x_n)$. Define the polynomial

$$f_G(\mathbf{x}) \triangleq \sum_{\{i,j\} \in E} x_i^{n_i(j)} x_j^{n_j(i)} + \sum_{j=1}^{n} x_j^{D_j}. \tag{2}$$

First, we check that a statement analogous to Claim 4.4 in [5] continues to hold for the f_G we have defined. The proof in [5] works for our f_G as well, we include a proof here for completeness.

▷ Claim 23 (Analogous to Claim 4.4 in [5]). For each $S \subseteq [n]$, we have $\mathrm{Rank}(M_S(f_G)) = |\mathrm{cut}(S)| + 2$.

Proof. The nonzero rows of $M_S(f_G)$ are of at most three types:
(1) Row indexed by the constant monomial 1;
(2) Rows indexed by $x_i^{n_i(j)}$ for $i \in S$ and $j \in [n] \setminus S$;
(3) Rows indexed by $x_i^{n_i(j)} x_j^{n_j(i)}$ for $i, j \in S$ or by $x_i^{D_i}$ for $i \in S$.
Similarly, columns of $M_S(f_G)$ are of at most three types:
(1) Column indexed by the constant monomial 1;
(2) Columns indexed by $x_j^{n_j(i)}$ for $j \in [n] \setminus$ and $i \in S$;
(3) Columns indexed by $x_i^{n_i(j)} x_j^{n_j(i)}$ for $i, j \in [n] \setminus S$ or by x_j^{Dj} for $j \in [n] \setminus S$.
Let $M_{i,j}$ denote the submatrix of $M_S(f_G)$ induced by rows of type (i) and columns of type (j) from the possibilities mentioned above. By construction, $M_{2,2}$ is a $|\mathrm{cut}(S)| \times |\mathrm{cut}(S)|$ permutation matrix. On the other hand, $M_{1,3}$ and $M_{3,1}$ are non-zero row and column matrices respectively (due to the presence of the $\sum x_j^{D_j}$ monomials in f_G). All the other submatrices $M_{i,j}$ are zero matrices. Therefore, $\mathrm{Rank}(M_S(f_G)) = \mathrm{Rank}(M_{2,2}) + \mathrm{Rank}(M_{1,3}) + \mathrm{Rank}(M_{3,1}) = |\mathrm{cut}(S)| + 2$. ◁

Forward Direction: If G has cutwidth at most w, then there exist $A \in \mathbf{GL}_n(\mathbb{F})$ and $\mathrm{b} \in \mathbb{F}^n$ such that $f_G(A\mathrm{x} + \mathrm{b})$ has an ROABP of width at most $w + 2$ in some order. We pick $A = I_n$ and $\mathbf{b} = 0$. By Claim 23, we have for each $S \subseteq [n]$, that $\mathrm{Rank}(M_S(f_G)) = |\mathrm{cut}(S)| + 2$. If there is a linear arrangement π that witnesses $\mathrm{cutwidth}(G) \leq w$, then by Nisan's characterization (Theorem 10), f_G has an ROABP in order π of width $\leq w + 2$.

Reverse Direction: If there exist $A \in \mathbf{GL}_n(\mathbb{F})$ and $\mathrm{b} \in \mathbb{F}^n$ such that $f_G(A\mathrm{x} + \mathrm{b})$ has an ROABP of width at most $w + 2$ in some order, then G has cutwidth at most w. We first prove the following key lemma which is the analogue of Lemma 20 in the case of characteristic p:

▶ **Lemma 24.** *Suppose $A \in \mathrm{GL}_n(\mathbb{F})$ is not the product of a permutation matrix and a diagonal matrix. Then for* every $\mathbf{b} \in \mathbb{F}^n$*, every ROABP computing $f_G(A\mathbf{x} + \mathbf{b})$ must have width at least $|E| + 3$.*

Proof. If A is not the product of a permutation matrix and a diagonal matrix, there is a row of A, say A_j, with at least two non-zero entries. Let j be the largest index for which this holds. The linear form $l_j(\mathbf{x}) = A_j\mathbf{x} + b_j$ has support at least 2. Assume, without loss of generality, that $l_j(\mathbf{x}) = a_{jk_1}x_{k_1} + a_{jk_2}x_{k_2} + \ldots$ with $k_1 \neq k_2$ and $a_{jk_1}, a_{jk_2} \neq 0$.

The polynomial $f_G(A\mathbf{x}+\mathbf{b})$ contains the term $(l_j(\mathbf{x}))^{D_j}$. By our choice of $D_j = (p^L-1)+(j-1)p^L$, Lucas's Theorem (Theorem 21) guarantees that $\binom{D_j}{i} \not\equiv 0 \pmod p$ for all $1 \le i \le M < D_j$. This ensures that in the expansion of $(l_j(\mathbf{x}))^{D_j}$, the coefficient of $x_{k_1}^i x_{k_2}^{D_j-i}$ is non-zero for all $1 \le i \le M-1$. Note that the support of these monomials is at least 2.

The other terms in $f_G(A\mathbf{x}+\mathbf{b})$ have lower total degree or have support at most 1: For $l > j$, the term $(A_l\mathbf{x}+b_l)^{D_l}$ only involves one variable. For $l < j$, the term $(A_l\mathbf{x}+b_l)^{D_l}$ has total degree $D_l < D_j$. The edge terms have total degree at most $6 < D_j$. Thus, the analysis is dominated by $(l_j(\mathbf{x}))^{D_j}$.

Now, consider any ROABP for $f_G(A\mathbf{x}+\mathbf{b})$ in an arbitrary order π. Pick a prefix S of π that separates k_1 and k_2. The width of the ROABP is at least $\mathrm{Rank}(M_S(f_G(A\mathbf{x}+\mathbf{b})))$. The submatrix of M_S corresponding to row monomials $\{x_{k_1}^i\}_{i=1}^{M-1}$ and column monomials $\{x_{k_2}^{D_j-i}\}_{i=1}^{M-1}$ is an anti-triangular matrix of size $(M-1)\times(M-1)$ with non-zero entries on its anti-diagonal. Its rank is therefore $M-1 \ge |E|+3$. Since π was arbitrary, this holds for any order. ◀

We use Lemma 24 to prove the reverse direction. Suppose there exist $A \in \mathrm{GL}_n(\mathbb{F})$ and $\mathbf{b} \in \mathbb{F}^n$ such that $f_G(A\mathbf{x}+\mathbf{b})$ has ROABP width at most $w+2$. Combining with Lemma 24 the reasoning provided in the proof of Theorem 19 we may assume that A is the identity matrix and that $\mathbf{b} = \mathbf{0}$. By Claim 23, we have

$$\text{cutwidth}(G) + 2 = \text{RO-width}(f_G(\mathbf{x})) \le w+2,$$

which implies $\text{cutwidth}(G) \le w$. This completes the soundness argument and the proof of the theorem. ◀

5 Hardness of Order-Finding for Quadratic Forms

The NP hardness reduction for ROABP order finding provided by [5] embeds the cut sizes of the graph into the ranks of the corresponding Nisan matrices. Instead, we can also embed the *cut-rank* information into the Nisan matrices. Let G be a graph and S be a subset of it's vertices. For a field $\mathbb{F}$, $\text{cut-rank}_{\mathbb{F}}(S)$ is defined as the rank over $\mathbb{F}$ of the $0-1$ matrix whose rows are indexed by vertices of G in S, columns by vertices not in S, and the (u,v)-th entry is 1 iff $\{u,v\}$ is an edge of G. This leads us to our next reduction, which gives NP-Hardness of order finding for quadratic forms. This is an improvement over [5], who give hardness for degree 6 polynomials).

▶ **Theorem 25.** *DenseROwidth-2 is NP-hard over any field $\mathbb{F}$.*

Proof. We reduce from the linear rank-width problem over $\mathbb{F}$. Let $(G=([n],E),w)$ be an instance of the linear rank-width problem. Construct the polynomial f_G over $\mathbb{F}$ defined as

$$f_G = \sum_{\{i,j\}\in E} x_i x_j + \sum_{i=1}^{n} x_i^2$$

The proof of NP-hardness follows from the next claim.

▷ Claim 26. Let $S \subseteq [n]$ be such that $1 \le |S| \le n-1$. Then $\mathrm{Rank}(M_S(f_G)) = \text{cut-rank}_{\mathbb{F}}(S)+2$

Proof. The proof follows by inspecting the structure of the matrix $M_S(f_G)$. Since f_G is a quadratic form, we only need to consider monomials of degree at most 2. We partition the rows and columns of the Nisan matrix $M_S(f_G)$ by the degree of the indexing monomials and obtain the following block structure for $M_S(f_G)$

$$M_S(f_G) \cong \begin{pmatrix} M_{0,0} & M_{0,1} & M_{0,2} \\ M_{1,0} & M_{1,1} & M_{1,2} \\ M_{2,0} & M_{2,1} & M_{2,2} \end{pmatrix}$$

Here, $M_{i,j}$ is the submatrix induced by row monomials of degree i and column monomials of degree j. First, note that $M_{1,1}$ is exactly the cut-rank matrix for the subset S. Also, note that $M_{2,0}$ and $M_{0,2}$ are non-zero (because of the x_i^2 terms) column and row matrices respectively, and so they have rank 1. Finally, note that the rest of the $M_{i,j}$ are all $\mathbf{0}$. Therefore, $\text{Rank}(M_S(f_G)) = \text{Rank}(M_{0,2}) + \text{Rank}(M_{2,0}) + \text{Rank}(M_{1,1}) = \text{cut-rank}_{\mathbb{F}}(S) + 2$ ◁

In particular, this implies that RO-width(f_G) is linear rank-width of G plus 2. ◀

References

1. Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM Journal on Computing*, 40(2):567–596, 2011. doi:10.1137/080729256.
2. Per Austrin, Toniann Pitassi, and Yu Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In Anupam Gupta, Klaus Jansen, José Rolim, and Rocco Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 13–24, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32512-0_2.
3. O. Baraskar, A. Dewan, C. Saha, and P. Sinha. NP-hardness of testing equivalence to sparse polynomials and to constant-support polynomials. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP)*, 2024.
4. Omkar Baraskar, Agrim Dewan, and Chandan Saha. Testing equivalence to design polynomials. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 9:1–9:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.9.
5. V. Bhargava, P. Dutta, S. Ghosh, and A. Tengse. The complexity of order-finding for roabps. arXiv preprint arXiv:2411.18981, 2024. arXiv:2411.18981.
6. Vishwas Bhargava and Sumanta Ghosh. Improved hitting set for orbit of roabps. *computational complexity*, 31(2):15, October 2022. doi:10.1007/s00037-022-00230-9.
7. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002. doi:10.1145/568522.568523.
8. Nikhil Gupta, Chandan Saha, and Bhargav Thankey. *Equivalence Test for Read-Once Arithmetic Formulas*, pages 4205–4272. SIAM, 2023. doi:10.1137/1.9781611977554.ch162.
9. Dori Medini and Amir Shpilka. Hitting sets and reconstruction for dense orbits in vp_{e} and ΣΠΣ circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 19:1–19:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.19.
10. N. Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 410–418, 1991.
11. S. Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017. doi:10.1016/J.DAM.2016.08.006.
12. Chandan Saha and Bhargav Thankey. Hitting sets for orbits of circuit classes and polynomial families. *ACM Trans. Comput. Theory*, 16(3), September 2024. doi:10.1145/3665800.

Additive, Near-Additive, and Multiplicative Approximations for APSP in Weighted Undirected Graphs: Trade-Offs and Algorithms

Liam Roditty
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Ariel Sapir[1]
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Abstract

We present a $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm for dense weighted graphs with a runtime of $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$, where W_i is the weight of an i^{th} heaviest edge on a shortest path between two vertices. Dor, Halperin and Zwick [FOCS'96 and SICOMP'00] introduced two algorithms for the commensurate unweighted $+2\cdot(k+1)$-APASP problem: one for sparse graphs with a runtime of $\tilde{O}\left(n^{2-\frac{1}{k+2}}m^{\frac{1}{k+2}}\right)$ and one for dense graphs with a runtime of $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$. Subsequently, Cohen and Zwick [SODA'97 and JALG'01] adapted the algorithm for sparse graphs to the weighted setting, namely a $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm with the same $\tilde{O}\left(n^{2-\frac{1}{k+2}}m^{\frac{1}{k+2}}\right)$ runtime. We fill the nearly three decades old gap by providing an algorithm for dense weighted graphs, matching the runtime for the unweighted setting.

In addition, we explore *nearly additive* APASP, where the multiplicative stretch is $1+\varepsilon$. We present a $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP algorithm with a runtime of $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W\right)$. This improves upon Saha and Ye [SODA'24], which had the same runtime, yet $(1+\varepsilon, 2W_1)$-APASP.

For pure multiplicative APASP, we present a $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm with a runtime of $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W\right)$. This improves, for dense graphs, the $\tilde{O}\left(nm^{\frac{2}{3}}+n^2\right)$ runtime of the $\frac{7}{3}$-APASP algorithm by Baswana and Kavitha [FOCS'06 and SICOMP'10], at the cost of introducing an additional ε to the multiplicative stretch.

We further view this result in a broader framework of $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP algorithms, similarly to the family of $\frac{3\ell+4}{\ell+2}$-APASP algorithms by Akav and Roditty [ESA'21]. This also generalizes the $(2+\varepsilon)$-APASP algorithm by Dory, Forster, Kirkpatrick, Nazari, Vassilevska Williams, and de Vos [SODA'24].

Finally, we show that it is possible to "bypass" an $\tilde{\Omega}(n^{\omega})$ conditional lower bound by Dor, Halperin, and Zwick for α-APASP with $\alpha<2$, by allowing an additive component to the approximation (e.g. a $\left(\frac{6k+3}{3k+2}, \sum_{i=1}^{k+1} W_i\right)$-APASP with $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ runtime.).

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Graph, Shortest Paths, Weighted Graphs, Approximation, Undirected, Single Source Shortest-Paths, Multi-Source Shortest-Paths, All-Pairs Shortest-Paths, SSSP, MSSP, MSASP, APSP, APASP

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.50

Related Version *Full Version*: http://www.arxiv.org/abs/2509.04640

[1] Corresponding author

© Liam Roditty and Ariel Sapir;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

We study the classic **A**ll-**P**airs **S**hortest **P**aths (*APSP*) problem in a weighted undirected graph $G = (V, E, w)$ with non-negative weights $w : E \to \mathbb{R}^+$. Floyd-Warshall's algorithm [30, 21, 28] solves APSP in $O(n^3)$ time. Johnson's algorithm [16] improves it to $O(n^2 \log n + nm)$, which is still $O(n^3)$ for dense graphs. Despite advances (e.g. [19, 24, 29]), no truly sub-cubic APSP algorithm currently exists. The fastest known one runs in $O\left(\frac{n^3}{2^{\sqrt{\Omega(\log n)}}}\right)$ time [29].

Facing this, an immediate question arises: *Can we approximate APSP in less than* $\tilde{O}(n^3)$ *time?* To explore this, we properly define the problem of **A**ll-**P**airs **A**pproximated **S**hortest **P**aths (*APASP*). For parameters α, β, the goal of an (α, β)-APASP is to compute, for all $u, v \in V$, an approximate distance $d[u, v]$ s.t. $\delta(u, v) \leq d[u, v] \leq \alpha \cdot \delta(u, v) + \beta$, where $\delta(u, v)$ is the distance between u and v. If $\beta = 0$, the approximation is purely multiplicative, referred to as an α-APASP, where α is the *multiplicative stretch*; if $\alpha = 1$, the approximation is purely additive, known as $+\beta$-APASP, where β is the *additive stretch*; otherwise, the approximation is "mixed".

In weighted graphs, defining additive approximation in a meaningful way is a bit tricky. By scaling edge weights, we can show that any constant $+\beta$-APASP is equivalent to APSP. To overcome this, we explore additive stretches that depend on w. One approach is a $+\beta W$-APASP, where $W = \max_{e \in E}\{w(e)\}$. However, it might hold that most shortest paths use edges whose weight is much smaller than W, hence a $+\beta W$-APASP yields a very "weak" guarantee.

Cohen and Zwick [9] addressed this issue by defining $W_i = W_i(u \rightsquigarrow v)$ to be the weight of an i^{th} heaviest edge on a shortest path $u \rightsquigarrow v$. This approach has since influenced many approximation results for various graphs distance problems (e.g., [22, 18, 1, 14]).

We consider the approach of Cohen and Zwick under a broader spectrum which generalizes the concept of equivalence between weighted and unweighted approximations. Consider an (α, β)-APASP in an unweighted graph for arbitrary α, β and let $f(\beta, G, P)$ be a function whose parameters are: a numerical value β, the weighted graph G and a path P between two vertices V. The problem of an $(\alpha, f(\beta, G, P))$-APASP in weighted graphs is a *commensurate version* of the (α, β)-APASP problem in unweighted graphs if, when the weight function $w(e) = 1$ for all $e \in E$, the additive stretch of the two problems is the same. That is, $f(\beta, G, P) = \beta$.

Two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, with runtimes $T_1(n)$ and $T_2(n)$, respectively, are *runtime equivalent* if $T_1(n) \in \tilde{\Theta}_W(T_2(n))$, where $\tilde{\Theta}_W(\cdot)$ hides poly-logarithmic factors of n or W. An $(\alpha, f(\beta, G, P))$-APASP algorithm $\mathcal{A}_1$ for weighted graphs is a *strongly commensurate version* of an (α, β)-APASP algorithm $\mathcal{A}_2$ in unweighted graphs if $(\alpha, f(\beta, G, P))$-APASP is a commensurate version of (α, β)-APASP and $\mathcal{A}_1$ and $\mathcal{A}_2$ are runtime equivalent. Thus, as long as $\mathcal{A}_2$ is not improved, $\mathcal{A}_1$ is optimal. We consider the broad question:

▶ **Question 1.** *For an (α, β)-APASP algorithm in unweighted graphs, what are its strongly commensurate versions $(\alpha, f(\beta, w))$-APASP algorithms in weighted graphs?*

Partial answers are already known. Cohen and Zwick [9] had two $+2W_1$-APASP algorithm with runtimes $\tilde{O}\left(n^{\frac{3}{2}} m^{\frac{1}{2}}\right)$ and $\tilde{O}\left(n^{\frac{7}{3}}\right)$, matching the runtimes of the two $+2$-APASP algorithms in unweighted graphs by Dor, Halperin, and Zwick [11]. Since $+2W_1 = +2$ when $w(e) = 1$ for all $e \in E$, the two pairs are, respectively, strongly commensurate.

Table 1 Our results for combinatorial and algebraic APASP for weighted graphs.

Approximation	Combinatorial	Algebraic	Reference
$+2\sum_{i=1}^{k+1} W_i$	$n^{2+\frac{1}{3k+2}}$	–	Section 4
$\frac{7}{3}+\varepsilon$	–	$mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\gamma)}\cdot\log W$ $\approx\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W$	Section 5
$\frac{3\ell+4}{\ell+2}+\varepsilon$	–	$mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\ell\cdot\gamma)}\cdot\log W$	Section 6
$(1+\varepsilon,\min\{2W_1,4W_2\})$	–	$mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\gamma)}\cdot\log W$ $\approx\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W$	Section 7

Cohen and Zwick also generalized one of their $+2W_1$-APASP algorithms to a $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm for weighted sparse graphs with $\tilde{O}\left(n^{2-\frac{1}{k+2}}m^{\frac{1}{k+2}}\right)$ runtime, hence finding a strongly commensurate version to the unweighted $+2\cdot(k+1)$-APASP algorithm of Dor, Halperin, and Zwick for sparse graphs. Yet, the $+2\cdot(k+1)$-APASP algorithm for unweighted dense graphs with $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ runtime of Dor, Halperin and Zwick was left unaddressed, raising the following question:

▶ **Question 2.** *Is there a $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm which is a strongly commensurate version of the $+2\cdot(k+1)$-APASP algorithm for dense unweighted graphs of [11]?*

Question 2 remained open for nearly three decades since the work of Cohen and Zwick [9], despite extensive progress on (α,β)-APASP algorithms for weighted graphs (e.g., [6, 17, 7, 3, 22, 12]). In this paper, we answer it in the affirmative by presenting a $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm for dense weighted graphs, running in $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ time – establishing a strongly commensurate version of Dor, Halperin, and Zwick's algorithm.

A meaningful observation is that, in this approximation, no edge should be taken into consideration more than once. That is, any i^{th} heaviest edge can contribute to the additive stretch at most once. Therefore, for $u,v\in V$, if there are less than k edges in a given shortest path $u\rightsquigarrow v$, the approximation which is due to that specfific path $u\rightsquigarrow v$, is bounded by $3\cdot\delta(u,v)$. If **all** shortest paths between u and v have less than k edges, the approximation for this pair is bounded by $3\cdot\delta(u,v)$.

In our approach, we utilized only hitting sets and **S**ingle **S**ource **S**hortest **P**aths (*SSSP*) invocations. However, other powerful tools exist. **M**atrix **M**ultiplication (*MM*), particularly **M**in **P**lus **M**atrix **M**ultiplication (*MPMM*), is a key tool for reducing the runtime of shortest path algorithms. Over the years, there have been many improvements (e.g. [5, 13, 27, 4]) on the exponent of n for the runtime of fast MM. However, these fast MM algorithms "hide" large constants in their runtime, which make them inefficient for practical use.

Motivated by this, Aingworth, Chekuri, Indyk and Motwani [2] initiated the study of algorithms for graph approximation problems (not necessarily APASP) that avoid fast MM. Since then, APASP algorithms have been broadly categorized into either: an *algebraic*

Table 2 Known additive (and nearly) combinatorial and algebraic APASP for unweighted graphs.

Approximation	Combinatorial	Algebraic
$+2$	$\min\left\{n^{\frac{7}{3}}, n^{\frac{3}{2}}m^{\frac{1}{2}}\right\}$ [11]	$n^2t^{\frac{1}{2}} + MPCRBD(n, \frac{n}{t})$ $\approx n^{2.2867}$ [10]
$(1+\varepsilon, 2)$	–	$n^{2+\frac{1-r}{2}} + \frac{1}{\varepsilon}n^{\omega(r)}$ $\approx \frac{1}{\varepsilon}\cdot n^{2.15135313}$ [12]
$+2\cdot(k+1)$	$\min\left\{n^{2+\frac{1}{3k+2}}, n^{2-\frac{1}{k+2}}m^{\frac{1}{k+2}}\right\}$ [11]	–

approach – which allows the use of fast MM (e.g., [8, 12, 17, 22, 31, 15]), and a *combinatorial approach* – where such algorithms are not used (e.g., [3, 6, 12, 17, 20, 9]). We summarize[2] results for unweighted graphs in Table 2 and for weighted graphs in Table 3.

One algebraic approach focuses on *nearly additive* approximations. Specifically, we show a $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP with a runtime of $\tilde{O}\left(\frac{1}{\varepsilon}\cdot n^{2.15135313}\cdot\log W\right)$. This improves upon Saha and Ye [22], which had the same runtime, yet computed a $(1+\varepsilon, 2W_1)$-APASP.

Another approach considers purely multiplicative stretches. We present a general framework of $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP algorithms with $\tilde{O}\left(mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\ell\cdot\gamma)}\cdot\log W\right)$ runtime, where β and γ are parameters which depend on the graph's density and ℓ is a parameter that indicates the quality of the stretch. When the graph is dense, the runtime becomes $\tilde{O}\left(n^{2+\beta}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-(\ell+1)\cdot\beta)}\cdot\log W\right)$, where $2+\beta=\omega(1-(\ell+1)\cdot\beta)$.

To properly understand the results that lie within this framework, we must first examine purely multiplicative stretches. In this setting, the goal is to compute an α-APASP as efficiently as possible. To do so, we first consider the known upper and lower bounds.

Dor, Halperin and Zwick [11] show that computing a $(2-\varepsilon)$-APASP, for any $\varepsilon>0$, is as hard as **B**oolean **M**atrix **M**ultiplication (BMM), which is only known to be solved in $\tilde{O}(n^{\omega})$[3]. This conditional lower bound means that 2-APASP is the smallest multiplicative stretch s.t. an α-APASP algorithm may require less than $\tilde{O}(n^{\omega})$ time.

Since APASP algorithms output an $n\times n$ matrix, $\Omega(n^2)$ time is unavoidable. For weighted graphs, the only known purely multiplicative stretch that can be computed in quadratic time is 3-APASP [9, 6]. This stands in contrast to the conditional lower bound for $(2-\varepsilon)$-APASP, implying the existence of a threshold $\alpha\in[2,3]$, for which there is an α-APASP algorithm with $\tilde{O}\left(n^2\right)$ runtime, while any $(\alpha-\varepsilon)$-APASP algorithm would require more[4] than quadratic time.

▶ **Question 3.** *What is the smallest $\alpha\in[2,3]$, for which there is an α-APASP algorithm that runs in $\tilde{O}\left(n^2\right)$ time, while any $(\alpha-\varepsilon)$-APASP algorithm necessitates $\tilde{\Omega}\left(n^{2+\gamma}\right)$ time?*

---

[2] Some results depend on certain parameters, such as $\omega(\alpha,\beta,\gamma)$ or $MPCRBD(n,m)$, which is the runtime for a MPMM on column and row bounded difference matrices of dimensions $n\times m$ and $m\times n$. We provide the mathematical expression and the numerical exponent with currently known values.

[3] $\omega(\alpha,\beta,\gamma)$ is the infimum of the exponent of the runtime for all algorithms which compute the multiplication of an $n^{\alpha}\times n^{\beta}$ and $n^{\beta}\times n^{\gamma}$ matrices. $\omega(\beta)=\omega(1,\beta,1)$ and $\omega=\omega(1)$.

[4] Assuming BMM cannot be solved in quadratic time.

Table 3 Known combinatorial and algebraic APASP runtimes for weighted graphs.

Approximation	Combinatorial	Algebraic
$+2W_2$	$n^{\frac{3}{2}}m^{\frac{1}{2}}$ [9]	-
$+\min\{2W_1, 4W_2\}$	$n^{\frac{7}{3}}$ [9]	-
$(1+\varepsilon, 2W_1)$	–	$\frac{1}{\varepsilon}\cdot n^{\omega(1,1-x,1)}\cdot\log W + n^{2+\frac{x}{2}}$ $\approx \frac{1}{\varepsilon}\cdot n^{2.15135313}\cdot\log W$ [22]
$+2\sum_{i=1}^{k+1} W_i$	$n^{2-\frac{1}{k+2}}m^{\frac{1}{k+2}}$ [9]	–
$\left(1+\varepsilon, 2\sum_{i=1}^{k+1} W_i\right)$	–	$\frac{1}{\varepsilon}\cdot n^{\omega\left(1-\frac{k}{k+2}\cdot x, 1-x, 1-\frac{k-1}{k+2}\cdot x\right)}\cdot\log W + n^{2+\frac{x}{k+2}}$ [22]
2	$n^{\frac{1}{2}}m + n^2$ [6, 3]	–
$2+\varepsilon$	–	$n^{1-r}m + \left(\frac{1}{\varepsilon}\right)^{O(1)} n^{\omega(r)}\cdot\log W$ $\approx \left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.21235201}\cdot\log W$ [12]
$(2, W_1)$	n^2 [6, 7]	–
$\frac{7}{3}$	$nm^{\frac{2}{3}} + n^2$ [6, 3]	–
$\frac{3\ell+4}{\ell+2}$	$n^{2-\frac{3}{\ell+2}}m^{\frac{2}{\ell+2}} + n^2$ [3]	–
3	n^2 [9, 6]	–

While exact quadratic-time algorithms are rare, there is notable progress for "near quadratic" ones, as summarized in Table 3. This paper contributes in this direction by presenting a $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm with $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W\right)$ runtime. For dense graphs, we improve the $\tilde{O}\left(m^{\frac{2}{3}}n + n^2\right)$ runtime of the $\frac{7}{3}$-APASP algorithm by Baswana and Kavitha [6], at the cost of adding an ε to the multiplicative stretch.

Baswana and Kavitha also presented a 2-APASP algorithm with $\tilde{O}\left(n^{\frac{3}{2}}m^{\frac{1}{2}} + n^2\right)$ runtime. Recently, Dory, Forster, Kirkpatrick, Nazari, Vassilevska Williams and de Vos [12] presented a $(2+\varepsilon)$-APASP algorithm with a $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.21235201}\cdot\log W\right)$ runtime, improving the runtime for dense graphs at the cost of adding an ε. This algorithm and our own $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm belong to a the previously discussed unified framework of $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP algorithms with $\tilde{O}\left(mn^{\beta} + n^{2+\gamma} + \left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\ell\cdot\gamma)}\cdot\log W\right)$ runtime.

While these results primarily explore trade-offs between the runtime and the multiplicative stretch, we would also like to fix the runtime and provide trade-offs between the multiplicative and additive stretches, proceeding in alignment with the direction of Saha and Ye [22].

We begin by focusing on cases where the multiplicative stretch is below 2, offering a way to "bypass" the conditional lower bound of $\Omega(n^{\omega})$ of Dor, Halperin and Zwick [11] for (α,β)-APASP algorithms where $\alpha < 2$.

▶ **Question 4.** *For which values of α, β such that $\alpha < 2$ is it possible to compute an (α,β)-APASP in less than $\tilde{O}(n^{\omega})$ runtime?*

We partially answer this by presenting a family of (α,β)-APASP algorithms with $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ runtime, such as $\left(\frac{6k+3}{3k+2}, \sum_{i=1}^{k+1} W_i\right)$-APASP. Furthermore, our approach helps bridge a known gap in the literature: no combinatorial algorithms currently exists for purely

Table 4 Runtime equivalence for $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$.

Approximation	Combinatorial	Reference
$+2\sum_{i=1}^{k+1} W_i$	$n^{2+\frac{1}{3k+2}}$	Section 4
$\left(\frac{6k+3}{3k+2}, \sum_{i=1}^{k+1} W_i\right)$	$n^{2+\frac{1}{3k+2}}$	Section 8
$\left(\frac{(9k+4)\cdot\alpha+(3k+2)\cdot\beta}{(\alpha+\beta)\cdot(3k+2)}, \frac{2\beta}{\alpha+\beta}\sum_{i=1}^{k+1} W_i\right)$	$n^{2+\frac{1}{3k+2}}$	Section 8
$\frac{9k+4}{3k+2}$	$\leq n^{2+\frac{1}{3k+2}}$	[3]

multiplicative α-APASP in weighted graphs where $\alpha \in \left(2, \frac{7}{3}\right)$. The lack of such can be partially accommodated by an (α, β)-APASP s.t. $\alpha \in \left(2, \frac{7}{3}\right)$ and β is small. For instance, we obtain a $\left(\frac{11}{5}, \frac{W_1+W_2}{4}\right)$-APASP in $\tilde{O}\left(n^{\frac{11}{5}}\right)$ time and a $\left(\frac{13}{6}, \frac{2\cdot(W_1+W_2+W_3)}{3}\right)$-APASP in $\tilde{O}\left(n^{\frac{17}{8}}\right)$ time (See: Appendix C). These belong to a wider family of approximations computable in $\tilde{O}(n^{2+\frac{1}{3k+2}})$ time (see Table 4). We broaden the question to any runtime function $T(n)$:

▶ **Question 5.** *Given an (α_1, β_1)-APASP with $T(n)$ runtime, what other (α_2, β_2)-APASP can be computed within $\tilde{O}(T(n))$ runtime?*

1.1 Related Work

Cohen and Zwick [9] had a 2-APASP algorithm with $\tilde{O}\left(n^{\frac{3}{2}}m^{\frac{1}{2}} + n^2\right)$ runtime and a $\frac{7}{3}$-APASP in $\tilde{O}\left(n^{\frac{7}{3}}\right)$ time. Both runtimes were improved by Baswana and Kavitha [6] to $\tilde{O}\left(m\sqrt{n} + n^2\right)$ and $\tilde{O}\left(m^{\frac{2}{3}}n + n^2\right)$, respectively. Kavitha [17] showed a $\frac{5}{2}$-APASP algorithm with $\tilde{O}\left(n^{\frac{9}{4}}\right)$ runtime, which was improved by Akav and Roditty [3] to $\tilde{O}\left(n^{\frac{5}{4}}m^{\frac{1}{2}} + n^2\right)$ time.

These multiplicative stretches were initially computed in an ad-hoc manner, each requiring its own tailored algorithm. The approaches for these APASP algorithms were developed without a unifying framework or a recognition of a broader underlying structure. Only later did Akav and Roditty [3] observe that these seemingly distinct stretches are, in fact, a part of a family of problems characterized by the form of $\frac{3\ell+4}{\ell+2}$-APASP that can be computed in $\tilde{O}\left(n^{2-\frac{3}{\ell+2}}m^{\frac{2}{\ell+2}} + n^2\right)$ time. For example, $\ell = 0$ indicates a 2-APASP algorithm that runs in $\tilde{O}\left(n^{\frac{1}{2}}m + n^2\right)$ and $\ell = 3$ indicates a $\frac{13}{5}$-APASP algorithm that runs in $\tilde{O}\left(n^{\frac{7}{5}}m^{\frac{2}{5}} + n^2\right)$ time.

A related problem is **M**ulti-**S**ource **S**hortest **P**aths (*MSSP*), where instead of computing all pairs of distances, we are given a set $S \subseteq V$ of sources and we need to compute the distances from this set to any $v \in V$. We define an (α, β)-MSASP similarly to an (α, β)-APASP. Let $\varepsilon > 0$, $|S| \in O(n^r)$ and assume edge weights are from $\{0, 1, \ldots, W\} \cup \{\infty\}$. Recently, Elkin and Neiman [15] a $(1+\varepsilon)$-MSASP algorithm with $\tilde{O}\left(m^{1+o(1)} + \left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{\omega(r)} \cdot \log W\right)$ runtime.

Dory, Forster, Kirkpatrick, Nazari, Vassilevska Williams and de Vos [12] utilized the result of Elkin and Neiman [15] and the 2-APASP of Baswana and Kavitha [6]. They achieved a $(2+\varepsilon)$-APASP algorithm with a $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{2.21235201} \cdot \log W\right)$ runtime, which improves

the runtime of the 2-APASP of Baswana and Kavitha [6] for dense graphs. We generalize this $(2+\varepsilon)$-APASP algorithm [12] and our own $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm, similarly to Akav and Roditty [3], and present a unified framework of $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP algorithms.

An equivalent approach was taken by Saha and Ye [22]. They computed a $(1+\varepsilon, 2\sum_{i=1}^{k+1} W_i)$-APASP. Their base case was $(1+\varepsilon, 2W_1)$-APASP with $\tilde{O}\left(\frac{1}{\varepsilon}\cdot n^{2.15135313}\cdot \log W\right)$ runtime. We improve the base case and compute a $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP, in the same runtime.

2 Toolkit

2.1 Basic graph notions

Unless stated otherwise, we consider an undirected graph $G=(V,E,w)$ with a non-negative weight function $w: E \to \mathbb{R}^{\geq 0}$, where the weight of an edge $(u,v)\in E$ is $w(u,v)$. Denote $|V|=n$ and $|E|=m$. For $k\in\mathbb{N}$, let $[k]=\{1,2,\ldots,k\}$. Each vertex $u\in V$ has its set of neighbours $N(u)$ and the set $\Gamma(u,k)$ of k-nearest neighbours. A *path* from a vertex u to a vertex v is an ordered list $u=y_0,y_1,y_2,\ldots,y_k,y_{k+1}=v$ s.t. for any $i\in[k+1]$: $(y_{i-1},y_i)\in E$. The weight of a path is $w(P)=\sum_{e\in P} w(e)$. A path with the smallest weight is a *shortest path* and its weight is the *distance*, denoted by $\delta(u,v)$. The distance between a vertex u to a set of vertices $S\subseteq V$ is the distance from u to the nearest $s\in S$. That is, $\delta(u,S)=\min_{s\in S}\{\delta(u,s)\}$. The *pivot* of u is a[5] vertex $p_S(u)$ such that $\delta(u,p_S(u))=\delta(u,S)$.

Denote a[6] shortest path between u and v by $u\rightsquigarrow v$: $w(u\rightsquigarrow v)=\delta(u,v)$. If $u\rightsquigarrow v$ is a single edge, we write u—v, hence $w(u,v)=\delta(u,v)$. Let $W_i(u\rightsquigarrow v)$ be the weight of the i^{th} heaviest edge. That is, $W_1(u\rightsquigarrow v)$ for the heaviest, $W_2(u\rightsquigarrow v)$ for the second heaviest, etc. The notion of a $W_i(u\rightsquigarrow v)$ applies to the fixed path $u\rightsquigarrow v$, and so does $\sum_{i\in I} W_i$. Yet, if no specific $u\rightsquigarrow v$ is stated, we consider **any**[7] shortest path, but all edges must be **from the same path**.

We denote by $d[u,v]$ the distance estimate computed by our algorithms. For an (α,β)-APASP algorithm, we prove that $\delta(u,v)\leq d[u,v]\leq \alpha\cdot\delta(u,v)+\beta$. Since G is undirected, we assume that any update to $d[u,v]$ also updates $d[v,u]$. This adds only $O(1)$ per update and does not affect the overall runtime.

2.2 Nearest vertices and pivots

Given several sets $T_1,T_2,\ldots,T_\ell$ of elements from a universe $U=\{u_1,u_2,\ldots,u_n\}$, a *hitting set* is a set $S\subseteq U$ such that for any $i\in[\ell]$: $S\cap T_i\neq\varnothing$. In our context, $U=V$ and $T_u=\Gamma(u,k)$ for $u\in V$ and some k that will be later specified. A hitting set, in this sense, is a set $S\subseteq V$ of small size which contains at least one of the k-nearest vertices of u, for all $u\in V$. Hence its pivot $p_S(u)$ is the nearest vertex between u and S. That is, $\delta(u,p_S(u))\leq\delta(u,s)$ for all $s\in S$.

Let $A\subseteq V$ and $u\in V$. We can consider $\Gamma(u,A,k)=\{a_{1,u},a_{2,u},\ldots,a_{k,u}\}$ where $\delta(u,a_{i-1,u})\leq\delta(u,a_{i,u})$ for $i\in[k]$, and for any $a\in A$: $\delta(a_{k,u})\leq\delta(a,u)$. Hence, the nearest vertices from the set A. These definitions are useful for APASP by distinguishing between

[5] If there are several candidates we can break ties by a consistent criteria.
[6] Several shortest paths may exist. If nothing is specifically mentioned, we consider an arbitrary one.
[7] Or, the smallest sum.

pairs of vertices $u, v \in V$ that are "far" from each other – and those that are "near" – that is, one lies within $\Gamma(\cdot, k)$ of the other. For near vertices, usually the exact distance is computed, by an algorithm [6, 25] similar to Dijkstra, while for "far" vertices, we use an approximation on paths that pass through the pivots.

$E_S(u) = \{(u, v) \mid w(u, v) < \delta(u, S)\}$ is the set of edges near u whose weight is smaller than $\delta(u, S)$, and $E_S = \bigcup_{u \in V} E_S(u)$. Denote the auxiliary edges $H_S = \{(u, p_S(u)) \mid u \in V\}$. If there are multiple hitting sets $H = \bigcup_S H_S$ is their union. When concerning a hierarchy of hitting-sets $S_1 \supseteq S_2 \supseteq \ldots$, we assume that $S_{i+1} \subseteq S_i$, and that each S_i has its set of edges E_{S_i} and pivots $p_i(u) \in S_i$ for each $u \in V$ (See Appendix A).

2.3 Min-plus matrix multiplication

A problem related to APSP is MPMM, which uses a different multiplication operator: given two matrices A and B, their product $C = A \star B$, satisfies $C[i, j] = \min_{k=1}^{n}\{A[i, k] + B[k, j]\}$. MPMM is runtime-equivalent to APSP [26]. However, MM and MPMM are not not known to be runtime-equivalent[8]. Nonetheless, faster MM algorithms can be utilized for MPMM by redefining the matrices (e.g. [23, 31, 8]).

Zwick [31] introduced **A**pproximated **M**in-**P**lus **M**atrix **M**ultiplication (*AMPMM*) in the same sense as APASP and presented a $(1 + \varepsilon)$-AMPMM algorithm with $\tilde{O}\left(\frac{1}{\varepsilon} \cdot n^{\omega} \cdot \log W\right)$ runtime. That is, this algorithm computes a matrix $\tilde{C}$, where $C[i, j] \leq \tilde{C}[i, j] \leq (1 + \varepsilon) \cdot C[i, j]$. It was later utilized for a $(1 + \varepsilon)$-APASP [8] in $\tilde{O}\left(\frac{1}{\varepsilon} \cdot n^{\omega} \cdot \log \frac{n}{\varepsilon}\right)$ time.

3 Overview

Common to all our algorithms is a hierarchy of hitting sets $V = S_0 \supseteq S_1 \supseteq \ldots \supseteq \varnothing$, where any $S_{i+1} \subseteq S_i$. Each level S_i has its own pivots $p_i(u)$ for every $u \in V$ and an associated edge set E_{S_i} (see: Section 2.2). The number and sizes of the hitting sets differs between the algorithms.

From the combinatorial perspective, our methods consist of selecting pivots through hitting sets and invoking multiple SSSP on graphs with sets of original and auxiliary sets of edges, tailored for the desired approximation. Some of our algorithms are order-sensitive considering the various SSSP invocations, as explicitly stated if so. We also consider distance updates through selected vertices, i.e. we have $d[u, x]$ and $d[x, v]$ and we update $d[u, v]$.

Our main contribution in the combinatorial setting is our $+2\sum_{i=1}^{k+1} W_i$-APASP algorithm for dense weighted graphs with a runtime of $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$. Extending the $+2 \cdot (k + 1)$-APASP algorithm of Dor, Halperin, and Zwick [11] to the weighted setting gives rise to several challenges. First, we use $\Gamma(u, k)$ instead of $N(u)$. This change is required to ensure that if there exists an edge $(x, y) \in u \rightsquigarrow v$ s.t. $(x, y) \notin E_{S_i}$, then $\delta(x, p_i(x)) \leq w(x, y)$, hence a path $u \rightsquigarrow p_i(x) \rightsquigarrow v$ would not "cost" us more than $2 \cdot w(x, y)$, which is at most $2W_1(u \rightsquigarrow v)$. Repeating this approach $k+1$ times results in an additive error of at most $2 \cdot (k + 1) \cdot W_1(u \rightsquigarrow v)$. However, our goal is to avoid reusing the weight W_1 of the heaviest edge repeatedly and to use the lighter edges $W_2, W_3, ..., W_{k+1}$. To obtain this, we must fix a certain shortest path and carefully make sure we do not use any edge more than once in the upper-bound for our

[8] If they would have been, as we already know $\omega < 3$, we could refute the APSP conjecture.

additive error. We achieve that by recursively concatenating auxiliary paths through pivots, ensuring that each additive term corresponds to a distinct edge from the original $u \rightsquigarrow v$. We emphasize that this issue does not exist in the unweighted setting, where the upper-bound is simply $+2 \cdot (k+1)$.

From the algebraic perspective, our methods consist of invoking $(1+\varepsilon)$-MSASP instead of several SSSP invocations, and $(1+\varepsilon)$-AMPMM for designated matrices. Note that rectangular MPMM can be used as a tool to replace distance updates through selected vertices.

Our main contribution in the algebraic setting is our unified framework of $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP algorithms with $\tilde{O}\left(mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{\omega(1-\beta-\ell\cdot\gamma)} \cdot \log W\right)$ runtime. For dense graphs $m \in \Theta\left(n^2\right)$, setting $\ell = 0$ results in the $(2+\varepsilon)$-APASP algorithm of Dory, Forster, Kirkpatrick, Nazari, Vassilevska Williams and de Vos [12], setting $\ell = 1$ results in our own $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm. For $\ell = 2$ we obtain a $\left(\frac{5}{2}+\varepsilon\right)$-APASP algorithm with $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{2.1185119} \cdot \log W\right)$ runtime. These results improve upon the framework of Akav and Roditty [3] for dense graphs, at the cost of an additional ε to the multiplicative stretch.

4 Additive $+2\sum_{i=1}^{k+1} W_i$-APASP

Algorithm 1 uses a parameter k that indicates a trade-off between the runtime and the approximation and parameters $\beta_1, \ldots, \beta_{3k+2} \in (0,1)$, whose values will be determined as part of the runtime analysis. Our algorithm works as follows: for each $i \in [3k+2]$ we set $\alpha_i = \sum_{j=1}^{i} \beta_j$ and compute a hitting set S_i to the sets $\Gamma(u, n^{\alpha_i})$, over all $u \in V$. For each S_i we also compute its corresponding E_{S_i}. We set $S_0 = V$ and $S_{3k+3} = \varnothing$, hence $E_{S_{3k+3}} = E$.

We initialize our distance approximation matrix $d : V \times V \to \mathbb{R}^{\geq 0}$ with edges' weights or ∞ otherwise. For $i \in [3k+2]$ and $u \in V$ we find the pivot $p_i(u)$ and set $d[u, p_i(u)] \leftarrow \delta(u, p_i(u))$ by a single SSSP invocation (See: Observation 14). We proceed with SSSP invocations from all $s \in S_i$ over the weighted graph $G_i(s) = \left(V, E_{S_{i+1}} \cup (\{s\} \times V) \cup H \cup F_i, d\right)$, where:

- $E_{S_{i+1}}$ - The union, over **all** $u \in V$, of edges (u, v) such that $w(u,v) < \delta(u, p_{i+1}(u))$,
- $\{s\} \times V$ - Auxiliary edges from s to V, representing distances previously computed,
- $H = \{(u, p_j(u)) \mid u \in V\}$ - Auxiliary edges from **any** u to its pivots $p_j(u)$ for $j \in [3k+2]$,
- F_i - Auxiliary edges from any $s_j \in S_j$ to any $s_\ell \in S_\ell$, for indices j, ℓ s.t. $i+j+\ell \geq 3k+1$.

The weights of these edges are given by $d : V \times V \to \mathbb{R}^{\geq 0}$, which is a weight function over $G_i(s)$. These SSSP invocations are performed in descending order, from $i \leftarrow 3k+2$ to 0, since they rely on auxiliary edges whose weights were previously computed. To properly understand the reason behind this, consider a $u \in V$ and let $i \in [3k+2]$ be the largest index such that $u \in S_i$. Whenever $d[p_{i+1}(v), u]$ is updated, we also update $d[u, p_{i+1}(v)]$. Therefore, when Algorithm 1 will invoke SSSP from $u \in S_i$, the auxiliary edge $(u, p_{i+1}(v))$ will be considered as well, and its weight is the already computed value $d[u, p_{i+1}(v)]$.

To prove an upper-bound of $+2\sum_{i=1}^{k+1} W_i$, we show $d[u,v] \leq \delta(u,v) + 2 \cdot (w(e_1) + \ldots + w(e_{k+1}))$ for **distinct** $k+1$ edges $e_1, \ldots, e_{k+1} \in u \rightsquigarrow v$. That is, our main argument would be that $e_i \neq e_j$, i.e.: no edge is taken into consideration in the upper-bound more than once. Let:

Algorithm 1 $+2\sum_{i=1}^{k+1} W_i$-APASP$(G=(V,E,w))$.

Input: An undirected non-negative weighted graph $G=(V,E,w)$ and $k \in \mathbb{N}$.
Output: A $+2\sum_{i=1}^{k+1} W_i$-APASP.

Let $\beta_1,\ldots,\beta_{3k+2} \in (0,1)$ to be fixed later
Initialize d as the edges' weights and ∞ otherwise
for $i \in [3k+2]$ **do**
  $\alpha_i \leftarrow \sum_{j=1}^{i} \beta_j$
  for $u \in V$ **do**
    Compute $\Gamma(u,V,n^{\alpha_i})$
  Compute S_i to be a hitting set to $\{\Gamma(u,V,n^{\alpha_i}) \mid u \in V\}$
  Compute the distances from any $u \in V$ to its pivot $p_i(u)$
  Construct the set of edges E_{S_i}
for $i \in \langle 3k+2,\ldots,1,0\rangle$ **do**
  $F_i \leftarrow \varnothing$
  for $j \in \{0,1,\ldots,3k+2\}$ **do**
    for $\ell \in \{3k+1-i-j, 3k+2-i-j,\ldots,3k+2\}$ **do**
      $F_i \leftarrow F_i \cup (S_\ell \times S_j)$
  for $s \in S_i$ **do**
    Invoke SSSP from s on $G_i(s) = \big(V, E_{S_{i+1}} \cup (\{s\}\times V) \cup H \cup F_i, d\big)$ and update d accordingly
return d

- $\kappa(u)$ be the largest index $i \in [3k+2]$ for which $u \in S_i$. As the graph is undirected, we can assume w.l.o.g that $\kappa(v) \le \kappa(u)$ (See: Figure 1).
- $\lambda(u \rightsquigarrow v)$ be smallest $j \in [3k+3]$ s.t. $u \rightsquigarrow v \subseteq E_{S_j} \cup H$, $u \rightsquigarrow v \not\subseteq E_{S_{j-1}} \cup H$ (See: Figure 2).
- $\xi(j, u \rightsquigarrow v)$ be the edge $(x,y) \in u \rightsquigarrow v$ for which $y \rightsquigarrow v \subseteq E_{S_{j+1}}$ and $(x,y) \notin E_{S_j} \cup H$. In other words, it is the first edge from v's side that does not belong to $E_{S_{j+1}} \cup H$. Additionally:
  - $\varphi(j, u \rightsquigarrow v) = x$ and $\psi(j, u \rightsquigarrow v) = y$ are the vertices of this edge (See: Figure 3).
  - $\zeta(j, u \rightsquigarrow v)$ is the maximal index $\ell \in [3k+2]$ for which: $\delta(y, p_\ell(y)) \le w(x,y)$. The edge $(x,y) \notin E_{S_{j+1}}$, hence $\delta(y, p_{j+1}(y)) \le w(x,y)$. Hence, such $j+1 \le \ell \le 3k+2$ exists.
- $\xi(u \rightsquigarrow v)$ be simply $\xi(\lambda(u \rightsquigarrow v)-1, u \rightsquigarrow v)$. That is, we view the first edge (x,y), from v's side, s.t. $(x,y) \notin E_{S_{\lambda(u \rightsquigarrow v)}} \cup H$ while $y \rightsquigarrow v \subseteq E_{S_{\lambda(u \rightsquigarrow v)}} \cup H$. Note such an edge exists due to the definition of $\lambda(u \rightsquigarrow v)$. We denote the endpoints of this edge by $\varphi(u \rightsquigarrow v)$ $\psi(j, u \rightsquigarrow v)$, and define: $\zeta(u \rightsquigarrow v) = \zeta(\lambda(u \rightsquigarrow v)-1, u \rightsquigarrow v)$ (Replace j by $\lambda(u \rightsquigarrow v)-1$ in Figure 3).
- $\Delta(u \rightsquigarrow v)$ be an upper-bound on the difference between the computed distance estimation $d[u,v]$ and the real distance $\delta(u,v)$, where $d[u,v]$ is computed due to the path $u \rightsquigarrow v$.
- $|u \rightsquigarrow v|$ be the number of edges in $u \rightsquigarrow v$.
- $\mu(x\text{—}y, u \rightsquigarrow v)$ be the[9] integer $j \in [|u \rightsquigarrow v|]$ such that $W_j(u \rightsquigarrow v) = w(x,y)$. In other words, it is an index specifying "how heavy" is the edge (x,y).

[9] If the weights of two edges on a certain shortest path are equal, we can break ties.

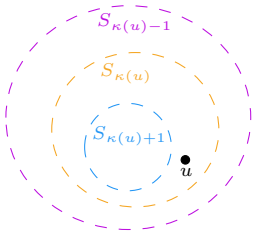


Figure 1 The (smallest hitting set) largest index $i \in [3k+2]$ s.t. $u \in S_i$.

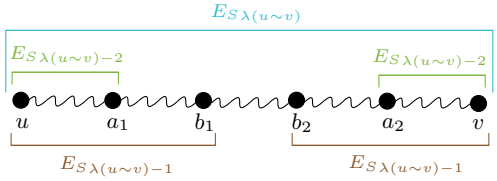


Figure 2 The first index $j \in [3k+2]$ s.t. $u \rightsquigarrow v \subseteq E_{S_j} \cup H$.

- $\mu(j, u \rightsquigarrow v)$ be equal to $\mu(\xi(j, u \rightsquigarrow v), u \rightsquigarrow v)$. That is, we ask "how heavy" is the edge $\xi(j, u \rightsquigarrow v)$, which is the first edge, from v's side, that is not in $E_{S_{j+1}} \cup H$.
- and $\mu(u \rightsquigarrow v)$ be $\mu(\xi(u \rightsquigarrow v), u \rightsquigarrow v)$ – "how heavy" is the edge $\xi(u \rightsquigarrow v)$, which is the first edge, from v's side, that is not in $E_{S_{\lambda(u \rightsquigarrow v)}} \cup H$.

We state a recurrence relation on $\Delta(u \rightsquigarrow v)$, considering three possible conditions:

▶ **Lemma 1.**

$$\Delta(u \rightsquigarrow v) \leq \begin{cases} 0 & \lambda(u \rightsquigarrow v) \leq \kappa(u)+1 \quad :(\text{A}) \\ \hline 2W_{\mu(u \rightsquigarrow v)} & \lambda(u \rightsquigarrow v) > \kappa(u)+1 \text{ and } \lambda(u \rightsquigarrow v) \geq 3k+1-\kappa(u)-\kappa(v) \quad :(\text{B}) \\ \hline 2W_{\mu(\kappa(u)+1, u \rightsquigarrow v)} + \Delta\left(p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\right) & \lambda(u \rightsquigarrow v) > \kappa(u)+1 \text{ and } \lambda(u \rightsquigarrow v) < 3k+1-\kappa(u)-\kappa(v) \quad :(\text{C}) \end{cases}$$

Where $x = \varphi(\kappa(u)+1, u \rightsquigarrow v)$ *and* $y = \psi(\kappa(u)+1, u \rightsquigarrow v)$.

The proof Lemma 1 is in Appendix B. The core of Algorithm 1's correctness lies within Lemma 2, where we utilize Lemma 1 and show two claims: One is that no edge is considered more than once in the upper-bound and the other is that there at most $k+1$ recurrences until we reach a halting condition – either Condition (A) or Condition (B) of Lemma 1. By proving both claims, we conclude that there are at most $k+1$ **distinct** edges $e_1, \ldots, e_{k+1} \in u \rightsquigarrow v$ that are considered in the upper-bound.

Fix a $u \rightsquigarrow v$. As G is undirected w.l.o.g. $\kappa(v) \leq \kappa(u)$. Let $(x, y) = \xi(\kappa(u)+1, u \rightsquigarrow v)$ be the first edge, from v's perspective, not in $E_{S_{\kappa(u)+1}}$. If we had an estimate $d[u, y]$, invoking SSSP from $u \in S_{\kappa(u)}$ would result in $d[u, v] \leq d[u, y] + \delta(y, v)$, as $y \rightsquigarrow v \subseteq E_{S_{\kappa(u)+1}}$.

To estimate $d[u, y]$, we would need an index $j \in [3k+2]$ s.t. $u \rightsquigarrow p_j(y) \rightsquigarrow y$ would be a good estimate on $\delta(u, y)$. To avoid having (x, y) considered more than once in the upper-bound, we select the pivot $p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y)$ as in Condition (C) of Lemma 1 (See: Figure 4).

As for the number of recurrences, note that Condition (A) and Condition (B) of Lemma 1 are the halting conditions of the recurrence relation. We prove two upper-bounds on the number of recurrences of Condition (C) of Lemma 1: one until reaching Condition (A) and the other until reaching Condition (B).

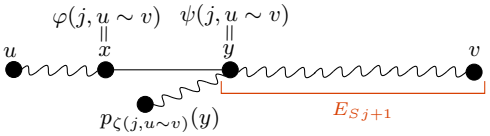


Figure 3 From v's perspective, the subpath $y \rightsquigarrow v \subseteq E_{S_{j+1}} \cup H$ while $(x, y) \notin E_{S_{j+1}} \cup H$. We denote $\varphi(j, u \rightsquigarrow v) = x$, $\psi(j, u \rightsquigarrow v) = y$ and $\xi(j, u \rightsquigarrow v) = (x, y)$. Note that $\delta(y, p_{j+1}(y)) \leq w(x, y)$, and let $\zeta(j, u \rightsquigarrow v)$ be the largest index s.t. $\delta\left(y, p_{\zeta(j,u\rightsquigarrow v)}(y)\right) \leq w(x, y)$.

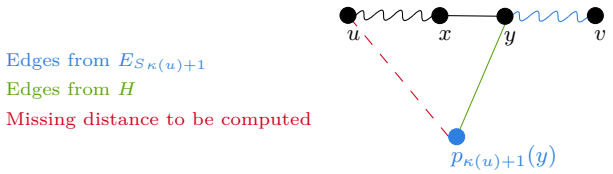


Figure 4 The difference Δ between d and δ. For a simpler, yet inaccurate, understanding, we illustrate $p_{\kappa(u)+1}(y)$. However, the correct pivot to consider is $p_{\zeta(\kappa(u)+1,u\rightsquigarrow v)}(y)$.

▶ **Lemma 2.** $\Delta(u \rightsquigarrow v) \leq 2 \sum_{i=1}^{k+1} W_i$

Proof. If we are either at Condition (A) or Condition (B) of Lemma 1, the claim is trivial. In Condition (C): $\Delta(u \rightsquigarrow v) = 2W_{\mu(\kappa(u)+1,u\rightsquigarrow v)} + \Delta\left(p_{\zeta(\kappa(u)+1,u\rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\right)$, where $x = \varphi(\kappa(u) + 1, u \rightsquigarrow v)$ and $y = \psi(\kappa(u) + 1, u \rightsquigarrow v)$. We want to show:

1. No edge is being taken into consideration in $\Delta(u \rightsquigarrow v)$ more than once.
2. The number of recurrences until (including) we reached a halting condition is at most $k + 1$.

We start by showing no edge is considered twice. Let $\hat{u} = p_{\zeta(\kappa(u)+1,u\rightsquigarrow v)}(y)$ and $\hat{v} = u$. Hence, in Condition (C): $\Delta\left(p_{\zeta(\kappa(u)+1,u\rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\right) = \Delta(\hat{u} \rightsquigarrow y — x \rightsquigarrow \hat{v})$. Observe that $\kappa(\hat{u}) = \kappa\left(p_{\zeta(\kappa(u)+1,u\rightsquigarrow v)}(y)\right) = \zeta(\kappa(u) + 1, u \rightsquigarrow v) \geq \kappa(u) + 1$. By the definition of $\zeta(\kappa(u) + 1, u \rightsquigarrow v)$ we know that $y — x \subseteq E_{S_{\zeta(\kappa(u)+1,u\rightsquigarrow v)+1}} = E_{S_{\kappa(\hat{u})+1}}$. Moreover, the added sub-path $\hat{u} \rightsquigarrow y \subseteq H$, as $\hat{u}$ is pivot of y. Consequently, $\hat{u} \rightsquigarrow y — x \subseteq E_{S_{\kappa(\hat{u})+1}} \cup H$. That is, the edge $\xi(\kappa(\hat{u}) + 1, \hat{u} \rightsquigarrow y — x \rightsquigarrow \hat{v})$ is from the path $x \rightsquigarrow \hat{v}$. Ergo, the recurrence in Condition (C) considers a different edge each time.

We now prove the second claim: for the number of recurrences of Condition (C). As we showed in each recurrence a different edge is considered, all that remains is to bound the number of edges, which is at most one more than the number of recurrences (if we halted due to Condition (B)). We consider two options: $\kappa(u) = \kappa(v)$ and $\kappa(u) > \kappa(v)$. We begin with the first, and by the end briefly consider the second option. We provide two bounds, one when the relation halted due to Condition (B) and the other when it halted due to Condition (A).

For Condition (B), observe that $\kappa(\hat{u}) \geq \kappa(u) + 1$ and $\kappa(\hat{v}) = \kappa(u) = \kappa(v)$. Let $\hat{\hat{u}} = p_{\zeta(\kappa(\hat{u})+1,\hat{u}\rightsquigarrow y — x\rightsquigarrow\hat{v})}(\psi(\kappa(\hat{u}) + 1, \hat{u} \rightsquigarrow y — x \rightsquigarrow \hat{v}))$ and $\hat{\hat{v}} = \hat{u}$. Therefore, $\kappa(\hat{\hat{u}}) \geq \kappa(\hat{u}) + 1$ and $\kappa(\hat{\hat{v}}) = \kappa(\hat{u}) \geq \kappa(u) + 1 > \kappa(u) = \kappa(\hat{v})$. In other words, in the first recurrence of Condition (C), $\kappa(\hat{u})$ increased by at least 1 compared to $\kappa(u)$ and $\kappa(\hat{v})$ is equal to $\kappa(v)$. By the second invocation, both $\kappa(\hat{\hat{u}})$ and $\kappa(\hat{\hat{v}})$ increase by at least 1 compared to $\kappa(\hat{u})$ and $\kappa(\hat{v})$, respectively. The value $\lambda(u \rightsquigarrow v) = \lambda(\hat{u} \rightsquigarrow y — x \rightsquigarrow \hat{v})$ because the only edges we add to

the path are from H and we neglect edges that belong to $E_{S_{\kappa(u)+1}} \subseteq E_{S_{\lambda(u \curlywedge v)-1}}$, assuming $\lambda(u \curlywedge v) > \kappa(u) + 1$. Therefore, there would be at most $\left\lceil \frac{3k+1-\lambda(u \curlywedge v)}{2} \right\rceil - \kappa(u)$ recurrences, which means at most $\left\lceil \frac{3k+1-\lambda(u \curlywedge v)}{2} \right\rceil - \kappa(u) + 1 < \frac{3k+1-\lambda(u \curlywedge v)}{2} - \kappa(u) + 2$ distinct edges.

For Condition (A), we bound the number of recurrences by $\lambda(u \curlywedge v) - \kappa(u) - 2$, as $\kappa(u)$ increases by at least 1 during each recurrence. This is also the number of distinct edges.

As both upper-bounds on the number of distinct edges hold, we compare them to find the actual bound: $\lambda(u \curlywedge v) - \kappa(u) - 2 = \frac{3k+1-\lambda(u \curlywedge v)}{2} - \kappa(u) + 2$. This holds for $\lambda(u \curlywedge v) = k + 3$. The number of edges is at most $\lambda(u \curlywedge v) - \kappa(u) - 2 = k + 1 - \kappa(u) \leq k + 1$, as $\kappa(u) \geq 0$.

As for the second option, $\kappa(u) \geq \kappa(v)$, the only change is that $\kappa(\hat{u})$ and $\kappa(\hat{v})$ increase by at least 1 from the first iteration rather than the second, which serves in our favor. ◀

▶ **Theorem 3.** *Algorithm 1 computes a* $+2\sum_{i=1}^{k+1} W_i$*-APASP in* $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ *time.*

The correctness proof and runtime analysis appear in the full version.

5 Multiplicative $\left(\frac{7}{3} + \varepsilon\right)$-APASP

Algorithm 2 uses two parameters $\beta, \gamma \in (0, 1)$ whose value will be determined when we will analyze the runtime. Our algorithm works as follows: we compute the hitting set S_1 (resp. S_2) for the sets $\Gamma\left(u, V, n^{\beta}\right)$ (resp. $\Gamma\left(u, V, n^{\beta+\gamma}\right)$), over all $u \in V$. We then compute the set $\Gamma(u, S_1, n^{\gamma})$. We compute E_{S_1} (resp. E_{S_2}). We also set $S_0 = V$ and $S_3 = \varnothing$.

We initialize our distance approximation matrix $d : V \times V \to \mathbb{R}^{\geq 0}$ to be either the weight of the edge or ∞. We find the pivots $p_1(u)$ (resp. $p_2(u)$) for all $u \in V$ and the exact distance $d[u, p_1(u)] \leftarrow \delta(u, p_1(u))$ (resp. $d[u, p_2(u)]$) by a single SSSP invocation (See: Observation 14). We then update distances for $(x, y) \in E$ to the pivots $p_1(v)$ and $p_2(v)$ of all $v \in \Gamma\left(y, V, n^{\beta}\right)$.

Let $i \in \{0, 1\}$. We begin by invoking SSSP from all $s \in S_i$ over the weighted graph $G_i(s) = \left(V, E_{S_{i+1}} \cup (\{s\} \times V) \cup H, d\right)$, where:

- $E_{S_{i+1}}$ - The union, over **all** $u \in V$, of edges (u, v) such that $w(u, v) < \delta(u, p_{i+1}(u))$,
- $\{s\} \times V$ - Auxiliary edges from s to V, representing distances previously computed,
- $H = \{(u, p_j(u)) \mid u \in V\}$ - Auxiliary edges from **any** u to its pivots $p_j(u)$ for $j \in \{1, 2\}$.

The weight of each such edge is given by $d : V \times V \to \mathbb{R}^{\geq 0}$, which is a weight function over $G_i(s)$. We then invoke a $(1 + \varepsilon)$-MSASP from S_2. In contrast to Algorithm 1, the order between the invocations of SSSP and MSASP here holds no importance.

Finally, we perform two distance updates. The first is $d[u, v] \leftarrow \min\{d[u, s] + d[s, v], d[u, v]\}$ for $s \in \Gamma(u, S_1, n^{\gamma})$ and the second is simply by all of the pivots of u and v, that is: $d[u, v] \leftarrow \min\{d[u, p_i(u)] + d[p_i(u), v], d[u, v]\}$.

To prove a multiplicative stretch of $\left(\frac{7}{3} + \varepsilon\right)$, we follow the steps of the proof of the $\frac{7}{3}$-APASP algorithm of Baswana and Kavitha [6] by providing a case analysis and proving each case individually.

▶ **Theorem 4.** *Algorithm 2 computes a* $\left(\frac{7}{3} + \varepsilon\right)$*-APASP in* $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{2.15135313} \cdot \log W\right)$ *time, for an arbitrary* $\varepsilon > 0$ *and weights of the set* $[W] \cup \{0, \infty\}$.

Algorithm 2 $\left(\frac{7}{3}+\varepsilon\right)$-APASP$(G=(V,E,w))$.

Input: An undirected non-negative weighted graph $G=(V,E,w)$ and an $\varepsilon>0$.
Output: A $\left(\frac{7}{3}+\varepsilon\right)$-APASP.

Let $\beta,\gamma\in(0,1)$ to be fixed later
Initialize d as the edges' weights and ∞ otherwise
Compute $\Gamma\left(u,V,n^{\beta}\right)$ (resp. $\Gamma\left(u,V,n^{\beta+\gamma}\right)$) for all $u\in V$
Compute S_1 (resp. S_2) as a hitting set to $\left\{\Gamma\left(u,V,n^{\beta}\right)\mid u\in V\right\}$ (resp. $\Gamma\left(u,V,n^{\beta+\gamma}\right)$)
Compute the distances from any $u\in V$ to its pivot $p_1(u)$ (resp. $p_2(u)$)
Construct the set of edges E_{S_1} (resp. E_{S_2})
for $(x,y)\in E$ **do**
  for $v\in\Gamma\left(y,V,n^{\beta}\right)$ **do**
    for $i\in\{0,1,2\}$ **do**
      $d[p_i(v),x]\leftarrow\min\{d[p_i(v),v]+d[v,y]+d[x,y],d[p_i(v),x]\}$
for $i\in\{0,1\}$ **do**
  for $s\in S_i$ **do**
    Invoke SSSP from s on $G_i(s)=\left(V,E_{S_{i+1}}\cup(\{s\}\times V)\cup H,d\right)$ and update d accordingly
Compute a $(1+\varepsilon)$-MSASP from S_2 on $G=(V,E,w)$ and update d accordingly
for $u,v\in V$ **do**
  for $s\in\Gamma(u,S_1,n^{\gamma})$ **do**
    $d[u,v]\leftarrow\min\{d[u,s]+d[s,v],d[u,v]\}$
for $u,v\in V$ **do**
  for $i\in[1,2]$ **do**
    $d[u,v]\leftarrow\min\{d[u,p_i(u)]+d[p_i(u),v],d[u,v]\}$
return d

In the full version, we prove the above and show how the runtime can be made to depend on the graph's density, namely:

$$\tilde{O}\left(mn^{\beta}+n^{2+\gamma}+\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\gamma)}\cdot\log W\right)$$

6 Multiplicative $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP

We extend our $\left(\frac{7}{3}+\varepsilon\right)$-APASP algorithm (Algorithm 2) into a framework of $\frac{3\ell+4}{\ell+2}$-APASP algorithms (Algorithm 3). Let ℓ be a parameter that indicates the trade-off between the runtime and the approximation and $\beta_1,\ldots,\beta_{\ell+1}\in(0,1)$ be parameters, whose values depend on graph's density. The proofs for Algorithm 3 appear in the full version.

Our algorithm works as follows: for $i\in[\ell+1]$ we set $\alpha_i=\sum_{j=1}^{i}\beta_j$ and compute a hitting set S_i to the sets $\Gamma(u,n^{\alpha_i})$, over all $u\in V$. We compute the corresponding E_{S_i}. We also set $S_0=V$ and $S_{\ell+2}=\varnothing$, hence $E_{S_{\ell+2}}=E$. We initialize the distance approximation matrix $d:V\times V\to\mathbb{R}^{\geq 0}$ to hold all edges' weights or ∞ else. We find the pivot $p_i(u)$ for all $u\in V$ and update the distance $d[u,p_i(u)]\leftarrow\delta(u,p_i(u))$ by a single SSSP invocation (See: Observation 14).

Algorithm 3 $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP$(G=(V,E,w))$.

Input: An undirected graph $G=(V,E)$, a positive weight function $w: E \to \mathbb{R}^+$, an integer $\ell \in \mathbb{N}$ and an arbitrary $\varepsilon > 0$.
Output: A matrix d which is a $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$-APASP.

Let $\beta_1, \ldots, \beta_{\ell+1} \in (0,1)$ to be fixed later
Initialize d as the edges' weights and ∞ otherwise
for $i \in [\ell]$ **do**
  $\alpha_i \leftarrow \sum_{j=1}^{i} \beta_j$
  for $u \in V$ **do**
    Compute $\Gamma(u, V, n^{\alpha_i})$
  Compute S_i to be a hitting set to $\{\Gamma(u, V, n^{\alpha_i}) \mid u \in V\}$
  Compute $\Gamma\left(u, S_i, n^{\beta_{i+1}}\right)$
  Compute the distances from any $u \in V$ to its pivot $p_i(u)$
  Construct the set of edges E_{S_i}
for $(x,y) \in E$ **do**
  for $i \in [\ell+1]$ **do**
    for $v \in \Gamma\left(y, S_{i-1}, n^{\beta_i}\right)$ **do**
      $d[p_i(v), x] \leftarrow \min\{d[p_i(v), v] + d[v,y] + d[x,y], d[p_i(v), x]\}$
for $i \in \{0\} \cup [\ell]$ **do**
  for $s \in S_i$ **do**
    Invoke SSSP from s on $G_i(s) = \left(V, E_{S_{i+1}} \cup (\{s\} \times V) \cup H, d\right)$ and update d accordingly
Compute a $(1+\varepsilon)$-MSASP from $S_{\ell+1}$ on $G=(V,E,w)$ and update d accordingly
for $u, v \in V$ **do**
  for $i \in [\ell]$ **do**
    for $s \in \Gamma\left(u, S_i, n^{\beta_{i+1}}\right)$ **do**
      $d[u,v] \leftarrow \min\{d[u,s] + d[s,v], d[u,v]\}$
for $u, v \in V$ **do**
  $\min_{i=0}^{\ell+1}\{d[u, p_i(u)] + d[p_i(u), v], d[u,v]\}$
return d

Following the initialization, we update the distances, for $i \in [\ell+1]$, from any $(x,y) \in E$ to $p_i(v)$ of $v \in \Gamma\left(y, S_{i-1}, n^{\beta_i}\right)$. That is, $d[p_i(v), x] \leftarrow \min\{d[p_i(v), v] + d[v,y] + d[x,y], d[p_i(v), x]\}$. We now consider an $i \in \{0\} \cup [\ell]$ and proceed with an SSSP invocation from all $s \in S_i$ over the weighted graph $G_i(s) = \left(V, E_{S_{i+1}} \cup (\{s\} \times V) \cup H, d\right)$, where:

- $E_{S_{i+1}}$ - The union, over **all** $u \in V$, of edges (u,v) such that $w(u,v) < \delta(u, p_{i+1}(u))$,
- $\{s\} \times V$ - Auxiliary edges from s to V, representing distances previously computed,
- $H = \{(u, p_j(u)) \mid u \in V\}$ - Auxiliary edges from **any** u to its pivots $p_j(u)$ for $j \in [\ell+1]$.

The weight of each is given by $d: V \times V \to \mathbb{R}^{\geq 0}$. Observe that d is a weight function over $G_i(s)$. We then proceed by a $(1+\varepsilon)$-MSASP invocation from the set of sources $S_{\ell+1}$ over the entire graph. We stress that, contrary to Algorithm 1 and similarly to Algorithm 2, the order of invocations of SSSP and MSASP here holds no importance.

Finally, we apply two types of distance updates. Let $u, v \in V$ and $i \in [\ell+1]$. First, $d[u,v] \leftarrow \min\{d[u,s]+d[s,v], d[u,v]\}$ through any $s \in \Gamma\big(u, S_i, n^{\beta_{i+1}}\big)$. The second is from all the $2\cdot(\ell+1)$ pivots[10] of both u and v, namely: $d[u,v] \leftarrow \min\{d[u,p_i(u)]+d[p_i(u),v], d[u,v]\}$.

To prove the correctness of our algorithm, we follow an analogous proof to the $\frac{3\ell+4}{\ell+2}$-APASP of Akav and Roditty [3]. As we performed changes to their algorithm, we provide a full proof of Algorithm 3, indicating where the additional ε affects. We begin with two upper-bounds:

▶ **Lemma 5.** $d[u,v] \le (1+\varepsilon)\cdot\min\{3\cdot\delta(u,v) - 2\cdot\delta(b,v), 3\cdot\delta(u,v) - 2\cdot\delta(u,a)\}$

▶ **Lemma 6.** $d[u,v] \le \delta(u,v) + 2\cdot(\ell+1)\cdot\delta(u,a)$ *or* $d[u,v] \le \delta(u,v) + 2\cdot(\ell+1)\cdot\delta(b,v)$

The proof of Lemma 5 and Lemma 6 appear in the full version. We follow with our main theorem:

▶ **Theorem 7.** *Let* $\alpha_j = \sum_{i=1}^{j}\beta_i$, $\varepsilon > 0$ *with weights from* $[W]\cup\{0,\infty\}$. *Algorithm 3 computes a* $\left(\frac{3\ell+4}{\ell+2}+\varepsilon\right)$*-APASP in* $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\alpha_{\ell+1})}\cdot\log W + mn^{\beta_1} + \sum_{i=2}^{\ell+1} n^{2+\beta_i}\right)$ *time.*

The proof for Theorem 7 appears in the full version. We observe that this runtime can be written as $\tilde{O}\Big(mn^{\beta} + n^{2+\gamma} + \left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{\omega(1-\beta-\ell\cdot\gamma)}\cdot\log W\Big)$, by selecting $\beta_1 = \beta$ and $\beta_2 = \ldots = \beta_{\ell+1} = \gamma$.

To perceive the wide range of our results, assume the graph is dense $m \in \tilde{\Theta}\big(n^2\big)$. When $\ell = 0$ we compute a $(2+\varepsilon)$-APASP in $\tilde{O}\Big(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.21235201}\cdot\log W\Big)$ time, which is the same as Dory, Forster, Kirkpatrick, Nazari, Vassilevska Williams, and de Vos [12]; when $\ell = 1$ our algorithm computes a $\left(\frac{7}{3}+\varepsilon\right)$-APASP in $\tilde{O}\Big(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.15135313}\cdot\log W\Big)$ time – as in Algorithm 2; when $\ell = 2$ our algorithm computes a $\left(\frac{5}{2}+\varepsilon\right)$-APASP in $\tilde{O}\Big(\left(\frac{1}{\varepsilon}\right)^{O(1)}\cdot n^{2.1185119}\cdot\log W\Big)$ time, etc.

7 Near additive $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP

Algorithm 4 has two parameters $\beta, \gamma \in (0,1)$ whose values depend on the graph's density and will be determined when we will analyze the runtime. It works as follows: we compute the hitting set S_1 (resp. S_2) for the sets $\Gamma\big(u, V, n^{\beta}\big)$ (resp. $\Gamma\big(u, V, n^{\beta+\gamma}\big)$), over all $u \in V$. We then compute the set $\Gamma(u, S_1, n^{\gamma})$. We compute E_{S_1} (resp. E_{S_2}). We also set $S_0 = V$ and $S_3 = \varnothing$.

The distance approximation matrix $d : V\times V \to \mathbb{R}^{\ge 0}$ is initialized to the weight of the edges or ∞ else-wise. We find the pivots $p_1(u)$ (resp. $p_2(u)$) for all $u \in V$ and the exact distance $d[u,p_1(u)] \leftarrow \delta(u,p_1(u))$ (resp. $d[u,p_2(u)]$) by a single SSSP invocation (See: Observation 14).

We proceed by invoking a $(1+\varepsilon)$-MSASP from S_2 over the entire graph. We then compute a $(1+\varepsilon)$-AMPMM of $d_0^2 \star d_2^0$, where d_i^j is a rectangular matrix containing all values of d, its rows narrowed down to include only the vertices of S_i and its columns to those of S_j. We note that similarly to Algorithm 1, the order is of importance here.

The following, in decreasing order $i \leftarrow 1$ to 0 (only two hitting sets), occurs twice: we invoke SSSP from all $s \in S_i$ on the graph $G_i(s) = \big(V, E_{S_{i+1}} \cup (\{s\}\times V) \cup H, d\big)$, where:

[10] Recall that $d[u,v]$ is updated whenever $d[v,u]$ is updated.

Algorithm 4 $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP$(G=(V,E,w))$.

Input: An undirected non-negative weighted graph $G=(V,E,w)$ and an $\varepsilon > 0$.
Output: A $(1+\varepsilon, \min\{2W_1, 4W_2\})$-APASP.

Let $\beta, \gamma \in (0,1)$ to be fixed later
Initialize d as the edges' weights and ∞ otherwise
Compute $\Gamma\left(u, V, n^{\beta}\right)$ (respectively, $\Gamma\left(u, V, n^{\beta+\gamma}\right)$) for all $u \in V$
Compute S_1 (respectively, S_2) to be a hitting set to $\left\{\Gamma\left(u, V, n^{\beta}\right) \mid u \in V\right\}$ (respectively, $\Gamma\left(u, V, n^{\beta+\gamma}\right)$)
Compute the distances from any $u \in V$ to its pivot $p_1(u)$ (respectively, $p_2(u)$)
Construct the set of edges E_{S_1} (respectively, E_{S_2})
Compute $(1+\varepsilon)$-MSASP from S_2 on $G=(V,E,w)$ and update d accordingly
Compute a $(1+\varepsilon)$-AMPMM $d_0^2 \star d_2^0$ and update d accordingly
for $j \in [2]$ **do**
  for $i \in \langle 1, 0\rangle$ **do**
    for $s \in S_i$ **do**
      Invoke SSSP from s on $G_i(s) = \left(V, E_{S_{i+1}} \cup (\{s\} \times V) \cup H, d\right)$ and update d accordingly
Compute a $(1+\varepsilon)$-AMPMM $d_0^1 \star d_1^1$ and update d accordingly
for $u \in S_0 = V$ **do**
  Invoke SSSP from u on $G_0(u) = (V, E_{S_1} \cup (\{u\} \times V) \cup H, d)$ and update d accordingly
return d

- $E_{S_{i+1}}$ - The union, over **all** $u \in V$, of edges (u,v) such that $w(u,v) < \delta(u, p_{i+1}(u))$,
- $\{s\} \times V$ - Auxiliary edges from s to V, representing distances previously computed,
- $H = \{(u, p_j(u)) \mid u \in V\}$ - Auxiliary edges from **any** u to its pivots $p_j(u)$ for $j \in \{1,2\}$.

In other words, after we performed an SSSP invocation from members of S_1, then $S_0 = V$, then S_1 and then again $S_0 = V$. Observe that $d : V \times V \to \mathbb{R}^{\geq 0}$ is a weight function over $G_i(s)$.

We then compute another $(1+\varepsilon)$-AMPMM, this time for the rectangular matrices $d_0^1 \star d_1^1$. Finally, we invoke one more SSSP from all $u \in V = S_0$ over the graph $G_i(u) = (V, E_{S_1} \cup (\{u\} \times V) \cup H, d)$.

To prove the *near additive* approximation of $(1+\varepsilon, \min\{2W_1, 4W_2\})$, we provide a case analysis and prove each case individually in the full version, where we also show the runtime analysis. We then conclude:

▶ **Theorem 8.** *Algorithm 4 runs in* $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{2.15135313} \cdot \log W\right)$ *time and computes a* $(1+\varepsilon, \min\{2W_1, 4W_2\})$*-APASP, for weights of the range* $[W] \cup \{0, \infty\}$.

We observe that while Algorithm 4 computes an equal or tighter approximation than that of Algorithm 2, the latter's runtime can be faster, as it depends on the graph's density.

8 Multiplicative and additive trade-offs

Recall Algorithm 1 which computed a $+2\sum_{i=1}^{k+1} W_i$-APASP and the $\frac{3\ell+4}{\ell+2}$-APASP algorithm by Akav and Roditty [3]. For dense graphs $m \in \tilde{\Theta}\left(n^2\right)$, the latter is a $\frac{9k+4}{3k+2}$-APASP with $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ runtime, by setting $\ell = 3k$.

▶ **Claim 9.** *Let $\alpha, \beta \in \mathbb{R}^+$. It is possible to compute a $\left(\frac{(9k+4)\cdot\alpha+(3k+2)\cdot\beta}{(\alpha+\beta)\cdot(3k+2)}, \frac{2\beta}{\alpha+\beta}\sum_{i=1}^{k+1} W_i\right)$-APASP in $\tilde{O}\left(n^{2+\frac{1}{3k+2}}\right)$ time.*

The proof is straightforward by running both algorithms (it appears in the full version). We observe that Claim 9 partially addresses Question 5 by presenting a family of APASP algorithms computable in $\tilde{O}(n^{2+\frac{1}{3k+2}})$ time. Claim 9 also partially answers Question 4, as, when $\alpha = \beta = 1$, we get a $\left(\frac{6k+3}{3k+2}, \sum_{i=1}^{k+1} W_i\right)$-APASP, whose multiplicative stretch is smaller than 2. For instance, $k = 1$ indicates that a $\left(\frac{9}{5}, W_1 + W_2\right)$-APASP can be computed in $\tilde{O}\left(n^{\frac{11}{5}}\right)$.

The same idea can be applied as well to Algorithm 3 that computes a $\left(\frac{3\ell+4}{\ell+2} + \varepsilon\right)$-APASP and the $\left(1+\varepsilon, 2\sum_{i=1}^{k+1} W_i\right)$-APASP algorithm of Saha and Ye [22], e.g. a $\left(\frac{5}{3}+\varepsilon, \min\{W_1, 2W_2\}\right)$-APASP computed in $\tilde{O}\left(\left(\frac{1}{\varepsilon}\right)^{O(1)} \cdot n^{2.15135313} \cdot \log W\right)$ time.

References

1. Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Stephen Kobourov, and Richard Spence. Weighted additive spanners. *arXiv*, 2021. arXiv:2002.07152.
2. Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.
3. Maor Akav and Liam Roditty. A unified approach for all pairs approximate shortest paths in weighted undirected graphs. In *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.4.
4. Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. *arXiv*, abs/2404.16349, 2024. doi:10.48550/arXiv.2404.16349.
5. Andris Ambainis, Yuval Filmus, and François Le Gall. Fast matrix multiplication: Limitations of the coppersmith-winograd method. *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, 2014. URL: https://api.semanticscholar.org/CorpusID:8332797.
6. Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010. doi:10.1137/080737174.
7. Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *Algorithms and Data Structures*, pages 541–552, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73951-7_47.
8. Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. Approximating apsp without scaling: equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 943–954, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316373.
9. Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38(2):335–353, 1997. doi:http://doi.org/10.1006/jagm.2000.1117.

10 Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. New additive approximations for shortest paths and cycles. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:10, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.50.

11 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. doi:10.1137/S0097539797327908.

12 Michal Dory, Sebastian Forster, Yael Kirkpatrick, Yasamin Nazari, Virginia Vassilevska Williams, and Tijn De Vos. Fast 2-approximate all-pairs shortest paths. *arXiv*, 2023. doi:10.1137/1.9781611977912.169.

13 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2129–2138, Los Alamitos, CA, USA, 2023. IEEE Computer Society. doi:10.1109/FOCS57990.2023.00130.

14 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Improved weighted additive spanners. In *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2021.21.

15 Michael Elkin and Ofer Neiman. Centralized, parallel, and distributed multi-source shortest paths via hopsets and rectangular matrix multiplication. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.27.

16 Donald Bruce Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, January 1977. doi:10.1145/321992.321993.

17 Telikepalli Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. *Algorithmica*, 63:224–245, 2012. doi:10.1007/s00453-011-9529-y.

18 An La and Hung Le. New weighted additive spanners. *arXiv*, 2024. doi:10.48550/arXiv.2408.14638.

19 Michael Lawrence Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976. doi:10.1137/0205006.

20 Liam Roditty. New algorithms for all pairs approximate shortest paths. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 309–320, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585197.

21 Bernard Roy. Transitivité et connexité. *Comptes rendus de l'Académie des Sciences*, 249:216–218, 1959. URL: https://gallica.bnf.fr/ark:/12148/bpt6k3201c/f222.image.

22 Barna Saha and Christopher Ye. *Faster Approximate All Pairs Shortest Paths*, pages 4758–4827. siam, 2023. doi:10.1137/1.9781611977912.170.

23 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. FOCS '99, page 605, USA, 1999. IEEE Computer Society.

24 Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992. doi:http://doi.org/10.1016/0020-0190(92)90200-F.

25 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52:1–24, 2005. doi:10.1145/1044731.1044732.

26 Alfred Vaino Aho, John Edward Hopcroft, and Jeffrey David Ullmand. *The Design and Analysis of Computer Algorithms*. Always learning. Pearson Education, 1974. URL: https://books.google.co.il/books?id=0ipKsZbwSAoC.

27 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. *New Bounds for Matrix Multiplication: from Alpha to Omega*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

28 Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, January 1962. doi:10.1145/321105.321107.

29 Richard Ryan Williams. Faster All-Pairs Shortest Paths via Circuit Complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

30 Robert Willoughby Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, June 1962. doi:10.1145/367766.368168.

31 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Detailed observations

In this section we provide several observations that are widely common for APASP algorithms that use hitting sets and pivots (e.g. [6, 25, 2]).

▶ **Observation 10** ([2]). *Let $r \in \mathbb{N}$ such that $|T_i| \geq r$ for any $i \in [\ell]$. A hitting set $S \subseteq U$ of size $\tilde{O}(\frac{n}{r})$ can be deterministically computed in $O(nr)$ time.*

Recall the notions mentioned in Section 2.1. While the following observations were proven for a very similar concept to $\Gamma(u,k)$, their proofs work for our case well.

▶ **Observation 11.** *Let $k \in [n]$. The sets $\Gamma(u,k)$ and the exact distances $d[u,x] = \delta(u,x)$ for any $x \in \Gamma(u,k)$ can be computed deterministically in $\tilde{O}(kn)$ time, for all $u \in V$.*

▶ **Observation 12** ([6]). *Let $u \in V$, $S \subseteq V$ and $v \in \Gamma(u,k)$. The edges of $u \rightsquigarrow v$ are contained within E_S.*

▶ **Observation 13** ([25]). *Let $u \in V$. Then: $|E_S(u)| \leq k$. Additionally, $|E_S| \leq nk$.*

It should be observed that finding the pivots themselves and the exact distances from each vertex to its pivot, can be accomplished by a single SSSP invocation with an auxiliary vertex connected to S with edges of weight 0.

▶ **Observation 14.** *Let $k \in [n]$. The pivot $p_S(u)$ of each vertex $u \in V$ and the exact distance $d[u, p_S(u)] = \delta(u, p_S(u))$ can be computed in $\tilde{O}(m)$ time.*

▶ **Observation 15.** *The set E_S can be computed in $\tilde{O}(nk)$ time.*

▶ **Observation 16.** *If $S_2 \subseteq S_1$ then $E_{S_1} \subseteq E_{S_2}$.*

B Proof of Lemma 1

In this section, we provide the missing proofs of Lemma 1 from Section 4. We recall:

▶ **Lemma 1.**

$$\Delta(u \rightsquigarrow v) \leq \begin{cases} 0 & \lambda(u \rightsquigarrow v) \leq \kappa(u)+1 \quad :(\text{A}) \\ 2W_{\mu(u \rightsquigarrow v)} & \lambda(u \rightsquigarrow v) > \kappa(u)+1 \text{ and } \lambda(u \rightsquigarrow v) \geq 3k+1-\kappa(u)-\kappa(v) \quad :(\text{B}) \\ 2W_{\mu(\kappa(u)+1, u \rightsquigarrow v)} + \Delta\big(p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\big) & \lambda(u \rightsquigarrow v) > \kappa(u)+1 \text{ and } \lambda(u \rightsquigarrow v) < 3k+1-\kappa(u)-\kappa(v) \quad :(\text{C}) \end{cases}$$

Where $x = \varphi(\kappa(u)+1, u \rightsquigarrow v)$ and $y = \psi(\kappa(u)+1, u \rightsquigarrow v)$.

Proof. Consider Condition (A), where $\lambda(u \rightsquigarrow v) \leq \kappa(u) + 1$. Hence, $S_{\kappa(u)+1} \subseteq S_{\lambda(u \rightsquigarrow v)}$. As $u \in S_{\kappa(u)}$, we invoke SSSP on $G_{\kappa(u)}(u)$ which includes the set of edges $E_{S_{\kappa(u)+1}}$. By Observation 16 we know that $E_{S_{\lambda(u \rightsquigarrow v)}} \subseteq E_{S_{\kappa(u)+1}}$. By the definition of $\lambda(u \rightsquigarrow v)$ we know that $u \rightsquigarrow v \subseteq E_{S_{\kappa(u)+1}}$. We conclude that the exact distance $d[u, v] = \delta(u, v)$ will be computed, resulting in $\Delta(u \rightsquigarrow v) = 0$.

If Condition (A) does not hold we know that $\kappa(u) < \lambda(u \rightsquigarrow v) - 1$. Recall that the SSSP invocations are performed in a decreasing order. Hence, when considering the SSSP invocation from either u or v, the distance estimations from $S_{\lambda(u \rightsquigarrow v)-1}$ to all the vertices have already been computed in a previous iteration.

We now move on to Condition (B), where additionally: $\lambda(u \rightsquigarrow v) \geq 3k + 1 - \kappa(u) - \kappa(v)$. Let us consider $i = \kappa(u)$, $j = \kappa(v)$ and $\ell = \lambda(u \rightsquigarrow v)$. Let $w = \varphi(u \rightsquigarrow v)$ and $z = \psi(u \rightsquigarrow v)$. Note that $w \rightsquigarrow p_{\lambda(u \rightsquigarrow v)-1}(w) \subseteq E_{S_{\lambda_{u \rightsquigarrow v}}}$ by definition, hence the concatenated path $p_{\lambda(u \rightsquigarrow v)-1}(w) \rightsquigarrow w — z \rightsquigarrow u, p_{\lambda(u \rightsquigarrow v)}(w) \rightsquigarrow w \rightsquigarrow v \subseteq E_{S_{\lambda_{u \rightsquigarrow v}}}$. Thus, the difference remains $\Delta\big(p_{\lambda(u \rightsquigarrow v)-1}(w) \rightsquigarrow w — z \rightsquigarrow u\big) = \Delta\big(p_{\lambda(u \rightsquigarrow v)-1}(w) \rightsquigarrow w \rightsquigarrow v\big) = 0$. In other words, $d\big[p_{\lambda(u \rightsquigarrow v)-1}(w), u\big] = \delta\big(p_{\lambda(u \rightsquigarrow v)-1}(w), u\big)$ and $d\big[p_{\lambda(u \rightsquigarrow v)-1}(w), v\big] = \delta\big(p_{\lambda(u \rightsquigarrow v)-1}(w), v\big)$. Recall that $\ell \geq 3k + 1 - i - j$, which means that when we invoke SSSP from u the edges $\{u\} \times S_\ell$ and $S_\ell \times S_j$ will be considered as well. Hence:

$$
\begin{aligned}
d[u, v] &\leq \underbrace{d\big[u, p_{\lambda(u \rightsquigarrow v)-1}(w)\big] + d\big[p_{\lambda(u \rightsquigarrow v)-1}(w), v\big]}_{\text{Due to SSSP from } u} = \underbrace{\delta\big(u, p_{\lambda(u \rightsquigarrow v)-1}(w)\big) + \delta\big(p_{\lambda(u \rightsquigarrow v)-1}(w), v\big)}_{\text{Our preceding discussion}} \\
&\leq \underbrace{\delta(u, w) + 2 \cdot \delta\big(w, p_{\lambda(u \rightsquigarrow v)-1}(w)\big) + \delta(w, v)}_{\text{Triangle inequality}} = \underbrace{\delta(u, v) + 2 \cdot \delta\big(w, p_{\lambda(u \rightsquigarrow v)-1}(w)\big)}_{u \rightsquigarrow w \rightsquigarrow v \text{ is a shortest path}} \\
&\leq \underbrace{\delta(u, v) + 2 \cdot w(z, w)}_{\text{By definition of } \lambda(u \rightsquigarrow v)} = \underbrace{\delta(u, v) + 2W_{\mu(u \rightsquigarrow v)}}_{\text{By definition of } \mu(u \rightsquigarrow v)}
\end{aligned}
$$

It follows that $\Delta(u \rightsquigarrow v) = d[u, v] - \delta(u, v) \leq 2W_{\mu(u \rightsquigarrow v)}$.

Finally, we are left with Condition (C), where we assume that $\lambda(u \rightsquigarrow v) < 3k + 1 - \kappa(u) - \kappa(v)$. In other words, we cannot use the edges through $p_{\lambda(u \rightsquigarrow v)-1}$, as invoking SSSP with these edges will simply take too much time. Let $x = \varphi(\kappa(u) + 1, u \rightsquigarrow v)$ and $y = \psi(\kappa(u) + 1, u \rightsquigarrow v)$, as stated in the lemma. Here, we note that if we had already previously computed $d\big[p_{\kappa(u)+1}(y), u\big]$, then an SSSP invocation from u can consider the following path:

1. Utilize the auxiliary edge $\big(u, p_{\kappa(u)+1}(y)\big)$, whose weight is simply $d\big[p_{\kappa(u)+1}(y), u\big]$.
2. Use the auxiliary edge $\big(p_{\kappa(u)+1}(y), y\big)$ whose weight is exactly $d\big[y, p_{\kappa(u)+1}(y)\big] = \delta\big(y, p_{\kappa(u)+1}(y)\big)$.
3. Continue on $y \rightsquigarrow v$, as $y = \psi(\kappa(u) + 1, u \rightsquigarrow v)$.

The weight of such a path is upper-bounded $d[u, v] \leq d\big[u, p_{\kappa(u)+1}(y)\big] + \delta\big(p_{\kappa(u)+1}(y), y\big) + \delta(y, v)$. Hence, $\Delta(u \rightsquigarrow v) \leq \Delta\big(p_{\kappa(u)+1}(y) \rightsquigarrow y — x \rightsquigarrow u\big) + 2 \cdot w(x, y)$. Recall that $\zeta(\kappa(u) + 1, u \rightsquigarrow v) \geq \kappa(u) + 1$ is defined to be the largest index for which the same edge (x, y) is at least as heavy as the distance to y's pivot. As a consequence, the above holds as well when instead of selecting $p_{\kappa(u)+1}(y)$, we select the pivot $p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y)$. Thus: $\Delta(u \rightsquigarrow v) \leq \Delta\big(p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\big) + 2 \cdot w(x, y)$.

The edge (x, y) is the first edge from v's side on $u \rightsquigarrow v$ that is not in $E_{S_{\kappa(u)+1}}$, which means that its the $\mu(\kappa(u) + 1, u \rightsquigarrow v)$ heaviest edge on the path $u \rightsquigarrow v$. In other words: $\Delta(u \rightsquigarrow v) \leq \Delta\big(p_{\zeta(\kappa(u)+1, u \rightsquigarrow v)}(y) \rightsquigarrow y — x \rightsquigarrow u\big) + 2 \cdot W_{\mu(\kappa(u)+1, u \rightsquigarrow v)}$. ◀

C Values for specified trade-offs

For completeness, we provide, in this section, values for the (α, β)-APASP examples where $\alpha \in \left(2, \frac{7}{3}\right)$ that were mentioned in Section 1 between Question 4 and Question 5. Consider the following:

- $\alpha = 3$, $\beta = 1$ and $k = 1$ results in $\left(\frac{11}{5}, \frac{W_1+W_2}{2}\right)$-APASP in $\tilde{O}\left(n^{\frac{11}{5}}\right)$ time;
- $\alpha = 2$, $\beta = 1$ and $k = 2$ results in $\left(\frac{13}{6}, \frac{2\cdot(W_1+W_2+W_3)}{2}\right)$-APASP in $\tilde{O}\left(n^{\frac{17}{8}}\right)$ time.

Randomized Black-Box PIT for Small Depth +-Regular Non-Commutative Circuits

G. V. Sumukha Bharadwaj
Department of Computer Science & Engineering, Indian Institute of Technology Tirupati, India

S. Raja
Department of Computer Science & Engineering, Indian Institute of Technology Tirupati, India

Abstract

In this paper, we address the black-box polynomial identity testing (PIT) problem for non-commutative polynomials computed by +-regular circuits, a class of homogeneous circuits introduced by Arvind, Joglekar, Mukhopadhyay, and Raja (STOC 2017, Theory of Computing 2019). These circuits can compute polynomials with a number of monomials that are doubly exponential in the circuit size. They gave an efficient randomized PIT algorithm for +-regular circuits of depth 3 and posed the problem of developing an efficient black-box PIT for higher depths as an open problem. Our work makes progress on this open problem by resolving it for constant-depth +-regular circuits.

We present a randomized black-box polynomial-time algorithm for +-regular circuits of any constant depth. Specifically, our algorithm runs in $s^{O(d^2)}$ time, where s and d represent the size and the depth of the +-regular circuit, respectively. Our approach combines several key techniques in a novel way. We employ a nondeterministic substitution automaton that transforms the polynomial into a structured form and utilizes polynomial sparsification along with commutative transformations to maintain non-zeroness. Additionally, we introduce matrix composition, coefficient modification via the automaton, and multi-entry outputs – methods that have not previously been applied in the context of black-box PIT. Together, these techniques enable us to effectively handle exponential degrees and doubly exponential sparsity in non-commutative settings, enabling polynomial identity testing for higher-depth circuits. In particular, we show that if f is a non-zero non-commutative polynomial in n variables over the field $\mathbb{F}$, computed by a depth-d +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and the size of the field $\mathbb{F}$ depends on the degree of f. Interestingly, the size of the matrices does not depend on the degree of f. Our result can be interpreted as an Amitsur-Levitzki-type result [2] for polynomials computed by small-depth +-regular circuits.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Polynomial Identity Testing, Non-commutative Circuits, Algebraic Circuits, +-Regular Circuits, Black-Box

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.51

Related Version *Full Version*: `https://arxiv.org/abs/2411.06569` [5]

Acknowledgements We would like to extend our sincere gratitude to Prof. Arvind (IMSc and CMI) for his valuable discussions. SR also expresses his sincere gratitude to Prof. Arvind and Prof. Meena Mahajan (IMSc) for facilitating a visit to IMSc, where part of this research was conducted. We also acknowledge the assistance of ChatGPT in rephrasing sections of this paper to improve clarity and articulation. However, we affirm that no technical ideas or proofs presented in this paper were generated by ChatGPT.

1 Introduction

The non-commutative polynomial ring, denoted by $\mathbb{F}\langle X\rangle$, over a field $\mathbb{F}$ in non-commuting variables X, consists of non-commuting polynomials in X. These are just $\mathbb{F}$-linear combinations of words (we call them monomials) over the alphabet $X = \{x_1, \ldots, x_n\}$. Hyafil [8]


© G. V. Sumukha Bharadwaj and S. Raja;
licensed under Creative Commons License CC-BY 4.0



and Nisan [10], studied the complexity of *non-commutative* arithmetic computations, in particular the complexity of computing the determinant polynomial with non-commutative computations.

Non-commutative arithmetic circuit families compute non-commutative polynomial families in a non-commutative polynomial ring $\mathbb{F}\langle X\rangle$, where multiplication is non-commutative (i.e., for distinct $x, y \in X$, $xy \neq yx$). Designing an efficient deterministic algorithm for non-commutative polynomial identity testing is a major open problem. Let $f \in \mathbb{F}\langle X\rangle$ be a polynomial represented by a non-commutative arithmetic circuit C. In this work, we assume that the polynomial f is given by a black-box access to C, meaning we can evaluate the polynomial f on matrices with entries from $\mathbb{F}$ or an extension field. Note that the degree of an n-variate polynomial computed by the circuit C of size s can be as large as 2^s and the sparsity, i.e., the number of non-zero monomials, can be as large as n^{2^s}. For example, the non-commutative polynomial $(x+y)^{2^s}$ has degree 2^s, doubly exponential sparsity 2^{2^s}, and has a circuit of size $O(s)$.

Bogdanov and Wee [6] have given an efficient randomized PIT algorithm for non-commutative circuits computing polynomials of degree $d = poly(s, n)$. Their algorithm is based on the result of Amitsur-Levitzki [2], which states the existence of matrix substitutions $M = (M_1, M_2, \ldots, M_n)$ such that the matrix $f(M_1, M_2, \ldots, M_n)$ is not the zero matrix, where the dimension of the matrices in M depends linearly on the degree d of the polynomial f. [2] shows that a non-zero non-commutative polynomial f of degree $2d-1$ does not vanish on the matrix algebra $\mathcal{M}_d(\mathbb{F})$. Since the degree of the polynomial computed by circuit C can be exponentially large in the size of the circuit, their approach will not work directly. Finding an efficient randomized PIT algorithm for general non-commutative circuits is a well-known open problem. It was highlighted at the Workshop on Algebraic Complexity Theory (WACT 2016) as one of the key problems to work on.

Recently, [3, 4] gave an efficient randomized algorithm for the PIT problem when the circuits are allowed to compute polynomials of exponential degree, but the sparsity (i.e., the number of non-zero monomials) could be exponential in the size of the circuit. In particular, it has been shown that if the sparsity of the polynomial is k, then identity testing can be performed using matrices of dimension $O(\log k)$. To handle doubly-exponential sparsity, they studied a class of homogeneous non-commutative circuits, that they call +-regular circuits, and gave an efficient deterministic white-box PIT algorithm. These circuits can compute non-commutative polynomials with the number of monomials that is doubly exponential in the circuit size. For the black-box setting, they obtain an efficient randomized PIT algorithm only for depth-3 +-regular circuits. In particular, they show that if a non-zero non-commutative polynomial $f \in \mathbb{F}\langle X\rangle$ is computed by a depth-3 +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_s(\mathbb{F})$ for a sufficiently large field $\mathbb{F}$. That is, the polynomial f cannot be identically zero under all substitutions from the matrix algebra $\mathbb{M}_s(\mathbb{F})$ for its variables. In other words, there exists at least one matrix substitution for the variables in f such that the resulting evaluated matrix is non-zero.

Finding an efficient randomized PIT algorithm for higher depth +-regular circuits is listed as an interesting open problem [3, 4]. We resolve this problem for constant depth +-regular circuits. In particular, we show that if $f \in \mathbb{F}\langle X\rangle$ is a non-zero non-commutative polynomial computed by a depth-d +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_N(\mathbb{F})$, with $N = s^{O(d^2)}$ and the size of the field $\mathbb{F}$ depends on the degree of polynomial f. We note that we get a *black-box* randomized polynomial time PIT algorithm for constant depth +-regular circuits.

1.1 Our results

The main result of the paper is the following theorem[1].

▶ **Theorem 1.** *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a +-regular circuit of depth d and size s. Then $f \not\equiv 0$ if and only if f is not identically zero on the matrix algebra $\mathbb{M}_N(\mathbb{F})$, with $N = s^{O(d^2)}$ and $\mathbb{F}$ is sufficiently large.*

For degree D non-zero non-commutative polynomial f, the classical Amitsur-Levitzki [2] theorem guarantees that f does not vanish on the matrix algebra $\mathcal{M}_{\frac{D}{2}+1}(\mathbb{F})$. If $D = 2^{\Omega(s)}$, this gives us an exponential time randomized PIT algorithm [6], where s is the size of the circuit computing f. In contrast, in our result, the dimension of the matrices is independent of the degree of the polynomial. If the sparsity of the polynomial, i.e., the number of non-zero monomials, is doubly exponential, then the main result of [4] gives only an exponential time randomized PIT algorithm as their matrix dimension depends on the logarithm of the sparsity.

This above theorem demonstrates that if the polynomial f is computed by a +-regular circuit of size s and depth $o(\sqrt{s}/\log s)$, we can determine if f is identically zero or not using a $2^{o(s)}$ time randomized PIT algorithm, which is exponentially faster than the existing methods. In particular, if depth is $O(1)$ then our algorithm runs in polynomial time. It is important to note that the number of product gates (within each Π^* layers) in any input-to-output path can be arbitrary and is only bounded by the circuit size s.

We note that [4] presented a white-box deterministic polynomial-time PIT for arbitrary depth +-regular circuits. For the small-degree case, [11] provided a white-box deterministic polynomial-time PIT for non-commutative ABPs, while [7, 1] have shown a deterministic quasi-polynomial-time black-box PIT algorithm for non-commutative ABPs. In the commutative setting, a randomized polynomial-time Polynomial Identity Test (PIT) was given by [9] using modular arithmetic evaluations.

1.2 Proof Sketch

We provide a brief sketch of our randomized PIT algorithm for constant-depth +-regular circuits. Our approach proceeds through a sequence of transformations:

1. A *structured transformation* using substitution automata, which converts the non-commutative polynomial into a more structured form while introducing spurious monomials.
2. A *product sparsification* step that reduces the number of non-commutative polynomial factors in each product to a small subset, preserving non-zeroness.
3. A *commutative transformation* step that reduces the PIT problem to the commutative setting.

Each step relies on small substitution automata and guarantees preservation of non-zeroness. These steps are carefully composed to preserve non-zeroness while keeping the dimension of the substitution matrices polynomial in s. We also introduce a novel coefficient modification strategy to prevent cancellation between structured and spurious terms.

In the rest of the paper, we will use "n.c." as an abbreviation for "non-commutative".

[1] For all omitted proofs, definitions, figures, and additional details, please refer to the full version of this paper [5]

1.3 Organization of the Paper

Section 2 presents our randomized black-box PIT algorithm for depth-5 +-regular circuits, highlighting the main steps: structure transformation, product sparsification, commutative transformation, and coefficient modification. Section 3 extends this result to arbitrary constant-depth +-regular circuits.

2 Black-Box PIT for $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ Circuits

In this section, we show that if $f \in \mathbb{F}\langle X\rangle$ is a n.c. polynomial of degree D computed by a depth-5 +-regular circuit of size s, then there exists an efficient randomized algorithm for identity testing the polynomial f. The main result of this section is the following theorem.

▶ **Theorem 2.** *Let f be a n.c. polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then $f \not\equiv 0$ if and only if f is not identically zero on the matrix algebra $\mathbb{M}_{s^6}(\mathbb{F})$.*

The polynomial f can be written as follows: $f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{ij}$. Here, the degree of each Q_{ij} is D_1, where $i \in [s], j \in [D_2]$, and it can be computed by a $\Sigma\Pi^*\Sigma$ circuit of size at most s. We establish Theorem 2 in three steps. First, we transform each polynomial Q_{ij} into a more structured n.c. polynomial, as defined below (see Definition 3). This structured polynomial has the property that we can simply consider it as a commutative polynomial preserving non-zeroness (Claim 5). As noted above, this is not true for n.c. polynomials, in general.

▶ **Definition 3.** *Let $s \in \mathbb{N} \cup \{0\}$. We call a n.c. polynomial g over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ as an s-ordered polynomial if it is of the form*

$$g = \sum_{i_1\geq 0,\ldots,i_s\geq 0} \alpha_{\bar{i}}.\xi_1^{i_1}\xi_2^{i_2}\ldots\xi_s^{i_s}, \text{ where } \alpha_{\bar{i}} \in \mathbb{F}.$$

▶ Remark 4. In the above definition, it is important to note that $i_j \geq 0$ for each $j \in [s]$. However, in this paper, we will focus on a special case of s-ordered polynomials, where $i_j > 0$, for each $j \in [s-1]$ and $i_s \geq 0$.

We have the following observation about the *s-ordered polynomial*.

▷ Claim 5. Suppose $g \in \mathbb{F}\langle\xi\rangle$ is an s-ordered polynomial. Let $g^{(c)}$ be the polynomial obtained by treating variables in ξ as commutative. Then, $g \not\equiv 0$ if and only if $g^{(c)} \not\equiv 0$.

▶ Remark 6. We will later allow coefficients $\alpha_{\bar{i}}$ to be commutative polynomials. The proof of this generalized statement follows the same reasoning as the proof of Claim 5.

Let us define the following concept, which will be relevant in subsequent sections:

▶ **Definition 7** (ξ-Pattern / Ordered Monomial)**.** *A ξ-pattern is a monomial of the form $\xi_1^{\ell_1}\xi_2^{\ell_2}\cdots\xi_k^{\ell_k}$, where $\ell_1, \ell_2, \ldots, \ell_k \geq 0$. In other words, it is a monomial in the variables $\xi_1, \xi_2, \ldots, \xi_k$ where the variables appear in the fixed order $\xi_1, \xi_2, \ldots, \xi_k$, but the exponents ℓ_i can vary. Such a monomial is also called an* ordered monomial.

We now explain the three steps of our method to establish Theorem 2.

2.1 Step 1: Transforming the Polynomial for Improved Structure

The initial step of our method involves transforming the polynomial to introduce more structure. During this process, we obtain a structured polynomial but also introduce some additional spurious monomials. This is one of the main differences between this work and [4]. We show that these spurious monomials have a distinguishing property that can be used to differentiate them from the structured part. We discuss the process of transforming each polynomial Q_{ij} into an s-ordered polynomial. To do this, we introduce a *new* set of commutative and n.c. variables as follows.
Let $Z = \{z_1, \ldots, z_n\}$ and let $Y = \{y_{ij} \mid i \in [n] \text{ and } j \in [s-1]\}$. The variables in Y and Z are commutative. Let $\xi = \{\xi_1, \xi_2, \cdots, \xi_s\}$ be the set of n.c. variables. We do this transformation using a small substitution automaton.

We consider the output of this substitution automaton on the n.c. polynomial f. The corresponding $s \times s$ substitution matrix $\mathbf{M_{x_i}}$ for each variable x_i is defined from this substitution automaton.

Output of the Automaton

Let $\mathbf{M} = f(\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$. Then we consider the *output of the automaton* as:

$$f' = \mathbf{M}[q_0, q_{s-1}] \tag{1}$$

which is a polynomial in the variables $\xi \sqcup Y \sqcup Z$.

Suppose a monomial m is computed by a $+$-regular circuit C. The monomial m has non-zero coefficient in $\prod_{j \in [D_2]} Q_{ij}$ for some $i \in [s]$. This monomial can be written as $m = m_1 \cdot m_2 \cdots m_{D_2}$, where each sub-monomial $m_j \in X^{D_1}$ has non-zero coefficient in Q_{ij}.

Next, we consider the output of the substitution automaton $\mathcal{A}$ on m. The automaton knows how to replace/substitute any variable x_j at any state q. Suppose $m = x_{i_1}.x_{i_2}.x_{i_3} \ldots x_{i_D}$, the output of the substitution automaton $\mathcal{A}$ on the monomial m is given by

$$\mathbf{M_m}[q_0, q_{s-1}]$$

where $\mathbf{M_m} = \mathbf{M_{x_{i_1}}} \cdot \mathbf{M_{x_{i_2}}} \cdots \mathbf{M_{x_{i_D}}}$. Each variable $x_i, i \in [n]$, is substituted by a degree two monomial over $\xi \sqcup Y \sqcup Z$ (one n.c. variable and one commutative variable). Consequently, the automaton transforms the monomial m into a degree $2D$ polynomial over $\xi \sqcup Y \sqcup Z$. Importantly, the *new* n.c. degree (i.e., sum of exponents of ξ variables) equals to the original degree D.

Computation by the substitution automaton

The automaton has exponentially many paths (with states allowed to repeat) from q_0 to q_{s-1}, all labeled by the same monomial m. Each computation path transforms the monomial m, originally over the variables X, into a new monomial over $\xi \sqcup Y \sqcup Z$.

For any path ρ from q_0 to q_{s-1}, we denote the transformed monomial as m_ρ. The polynomial computed by $\mathbf{M_m}[q_0, q_{s-1}]$ is given by

$$\sum_{\rho : q_0 \overset{m}{\rightsquigarrow} q_{s-1}} m_\rho,$$

which is a polynomial in $\mathbb{F}[Y \sqcup Z]\langle\xi\rangle$. Recall that n.c. polynomials are $\mathbb{F}$-linear combinations of words/strings (called monomials). For a n.c. monomial m, we can identify the variable at position e in m, where $1 \leq e \leq |m|$.

Recall that m can be written as $m = m_1 \cdot m_2 \cdots m_{D_2}$. Each computation path ρ substitutes each n.c. variable in m according to the automaton's transition rules, resulting in a monomial m_ρ over new variables $\xi \sqcup Y \sqcup Z$. We group all commutative variables appearing in m_ρ and denote it by c_m, which is a commutative monomial over $Y \sqcup Z$. The resulting monomial m_ρ has the following form.

▶ **Proposition 8.** *Let ρ be a path from q_0 to q_{s-1} labeled by the monomial m. The transformed monomial m_ρ can be expressed in the form: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$, where $N \geq 1$, c_m is a monomial over $Y \sqcup Z$, and each m'_ℓ (for $\ell \in [N]$) is given by $m'_\ell = \xi_1^{\ell_1} . \xi_2^{\ell_2} \cdots \xi_s^{\ell_s}$, where $\ell_k > 0$ for $k \in [s-1]$, and $\ell_s \geq 0$.*

For $i \neq j$, the exponents of the ξ variables in the sub-monomials m'_i and m'_j can vary. In particular, it is generally possible that $(i_1, i_2, \ldots, i_s) \neq (j_1, j_2, \ldots, j_s)$.

Types of sub-monomials: Two cases

It is important to note that the number of new sub-monomials m'_i, denoted as N, may not be equal to D_2. This is because N depends on how many times the path ρ returns to the initial state q_0. Also, the sum of exponents of ξ variables in each sub-monomial m'_i in m_ρ can vary. This leads us to consider two possible cases for each computation path ρ that starts at q_0 and ends at q_{s-1}: (recall $m = m_1 \cdot m_2 \cdots m_{D_2}$).

- **Case 1:** For each $j < D_2$, the boundary between m_j and m_{j+1} in m is respected by the path ρ. In this computation path ρ, the state of $\mathcal{A}$ is at q_0 precisely when it begins processing each sub-monomial $m_j \in X^{D_1}$ for $j \in [D_2]$. This means that when $\mathcal{A}$ reads the last variable of the sub-monomial m_{j-1} (for $j > 1$), it transitions back to state q_0. As a result, $\mathcal{A}$ is in state q_0 exactly when it reads the first variable of the sub-monomial m_j. This holds true for all sub-monomials m_j where $j \in [N]$. By Proposition 8, the transformed monomial can be expressed as: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$ where c_m is a monomial over $Y \sqcup Z$ and each m'_ℓ is of form $m'_\ell = \xi_1^{\ell_1} \cdots \xi_s^{\ell_s}$. In this case, we observe that $N = D_2$ since there are exactly D_2 sub-monomials in m. *or*
- **Case 2:** For some $j < D_2$, the boundary between m_j and m_{j+1} in m is not respected by the path ρ. In this case, there exists a sub-monomial m_j, where $j \in [D_2]$, such that either (1) the computation path ρ visits the state q_0 while processing the variable located at position c, where $1 < c \leq D_1$. This means ρ returns to q_0 in the middle of processing m_j. *or* (2) the path ρ is in a state $q_j, j \neq 0$ (i.e., other than the initial state q_0) while processing the variable that appears at the first position of the sub-monomial m_j. By Proposition 8, the transformed monomial can be expressed as: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$ where c_m is a monomial over $Y \sqcup Z$ and each m'_ℓ is of form $m'_\ell = \xi_1^{\ell_1} \cdots \xi_s^{\ell_s}$. In this case, we cannot definitively say whether N is equal to D_2 or not.

▶ Remark 9. Any path ρ from q_0 to q_{s-1} labeled by a monomial $m \in X^D$ will satisfy either Case 1 or Case 2, but not both.

In Case 1, we can make the following important observation about the obtained monomial m_ρ. Recall that D_1 is the degree of Q_{ij} polynomial for all $i \in [s]$ and $j \in [D_2]$.

▷ Claim 10. Let ρ be a path from q_0 to q_{s-1} labeled by the monomial m that satisfies Case 1. In this case, for each sub-monomial m'_ℓ, where $\ell \in [D_2]$, of the monomial m_ρ, the sum of the exponents of its n.c. variables is D_1. That is, $\sum_{j \in [s]} \ell_j = D_1$.

For all paths ρ that satisfy Case 2, this is not true. We note this down as the following claim.

▷ Claim 11. Let ρ be a path from q_0 to q_{s-1} labeled by the monomial m that satisfies Case 2. In this case, there exists a sub-monomial m'_ℓ, where $\ell \in [N]$, in the obtained monomial m_ρ such that the sum of the exponents of its n.c. variables is *not* equal to D_1. That is, $\sum_{j\in[s]} \ell_j \neq D_1$.

We crucially utilize Claims 10 and 11 later to ensure the non-zeroness of the transformed commutative polynomial.

The *structured* part and the *spurious* part

For a monomial $m = m_1 \cdot m_2 \cdots m_{D_2}$, we define the polynomial $\hat{f}_m$ as the sum of all monomials that are obtained from computation paths ρ labeled by m from Case 1 above. Similarly, the polynomial F_m is defined as the sum of all monomials obtained from computation paths ρ labeled by m from Case 2 above. We consider the output of the substitution automaton $\mathcal{A}$ on the given n.c. polynomial $f \in \mathbb{F}\langle X\rangle$. The output of the automaton is the sum of all monomials produced by computation paths ρ starting from q_0 and leading to q_{s-1}, with these paths labeled by monomials generated by a depth-5 +-regular circuit.

Let $Mon(f)$ be the set of all monomials computed/generated by the given depth-5 +-regular circuit computing f. That is, suppose m is computed by $\prod_{j\in[D_2]} Q_{i,j}$ for some $i \in [s]$, with coefficient $\alpha_{m,i}$ then $\alpha_{m,i} \cdot m \in Mon(f)$. Let

$$\hat{f} = \sum_{\alpha_{m,i}\cdot m \in Mon(f)} \hat{f}_{\alpha_{m,i}\cdot m} \tag{2}$$

$$F = \sum_{\alpha_{m,i}\cdot m \in Mon(f)} F_{\alpha_{m,i}\cdot m}. \tag{3}$$

We refer to F as the sum of spurious monomials obtained from the automaton, which can be viewed as noise resulting from our method.

We assume that the linear forms in the $Q_{i,j}$ polynomials are numbered from 1 to D_1. For $I \subseteq [D_1]$ with size at most $s-1$, define $Q_{i,j,I}$ as the polynomial obtained from $Q_{i,j}$ by treating linear forms indexed by I as non-commuting and the rest of the linear forms as commuting. We also substitute the n.c. variables that appear in linear forms indexed by I with double-indexed commutative variables Y, as shown in the substitution automaton.

We have the following claim regarding the polynomial $\hat{f}$.

▷ Claim 12. The polynomial $\hat{f}$ can be expressed as $\hat{f} = \sum_{i\in[s]} \prod_{j\in[D_2]} \sum_{I\subseteq[D_1],|I|=s-1} Q_{i,j,I} \times \xi_I$ where $\xi_I = \xi_1^{\ell_1} \cdot \xi_2^{\ell_2-\ell_1} \cdots \xi_s^{D-\ell_{s-1}}$ for $I = \{\ell_1, \ell_2, \cdots, \ell_{s-1}\}$ such that $\ell_1 < \ell_2 < \cdots < \ell_{s-1}$, and $Q_{i,j,I}$ is the polynomial obtained from $Q_{i,j}$ by treating the linear forms indexed by I as non-commuting and the rest of the linear forms as commuting.

Let

$$\hat{Q}_{ij} = \sum_{I\subseteq[D_1],|I|=s-1} Q_{i,j,I} \times \xi_I. \tag{4}$$

Then we can express $\hat{f}$ as:

$$\hat{f} = \sum_{i\in[s]} \left(\prod_{j\in[D_2]} \hat{Q}_{ij} \right). \tag{5}$$

The output of the substitution automaton $\mathcal{A}$ on the polynomial f is given by:

$$f' = \hat{f} + F.$$

This is stated in the following claim.

▷ Claim 13. Let f be a homogeneous n.c. polynomial computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then, the output $f' \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ of the substitution automaton $\mathcal{A}$ on the polynomial f can expressed as $f' = \hat{f} + F$, where $\hat{f}$ is the structured part as defined in Equation 5 and F is the spurious part as defined in Equation 3.

2.1.1 Non-zeroness of $\hat{f}$

We establish that f' is non-zero by first proving that $\hat{f}$ is not zero. This is shown in Lemma 15, which builds on the result of PIT for $\Sigma\Pi^*\Sigma$ circuits (see Section 6.2 in [4]). We briefly discuss this result.

Let $Z = \{z_1, \ldots, z_n\}$ be the set of *new* commuting variables. Let $g \in \mathbb{F}\langle X\rangle$ be a polynomial of degree D computed by a $\Sigma\Pi^*\Sigma$ circuit of size s. Then g can be expressed as $g = \sum_{i\in[s]} \prod_{j\in[D]} L_{ij}$, where L_{ij} are homogeneous linear forms. Let $P_i = \prod_{j\in[D]} L_{ij}$, $i \in [s]$. We have $g = \sum_{i\in[s]} P_i$. For $I \subseteq [D]$ with size at most $s-1$, define $P_{i,I}$ as the polynomial obtained from P_i by treating linear forms indexed by I as non-commuting and the rest of the linear forms as commuting. We replace each n.c. variable x_i appearing in $[D] \setminus I$ by a *new* commuting variable z_i.

The number of n.c. linear forms appearing in $P_{i,I} \in \mathbb{F}[Z]\langle X\rangle$ is bounded by $|I| < s$. This is because the linear forms that appear with indices other than those in I are treated as commutative. Consequently, the number of non-commutative linear forms in $P_{i,I} \in \mathbb{F}[Z]\langle X\rangle$ is bounded by $|I| < s$. We refer to this as the n.c. degree of the polynomial $P_{i,I}$. Since this degree is small, $P_{i,I}$ can be converted into a commutative polynomial while preserving its non-zeroness. Let $P^{(c)}_{i,I}$ denote the commutative polynomial obtained from $P_{i,I}$ and define $g_I = \sum_{i\in[s]} P^{(c)}_{i,I}$. To keep all guesses of the set I distinct, additional commutative variables $\xi = \{\xi_1, \xi_2, \cdots, \xi_{k+1}\}$ are introduced in [4]. The transformed commutative polynomial obtained in [4] is given by:

$$g^\star = \sum_{I\subseteq[D_1], |I|=k} g_I \times \xi'_I \tag{6}$$

where $\xi'_I = \xi_1^{\ell_1-1}.\xi_2^{\ell_2-\ell_1-1}\cdots\xi_{k+1}^{D-\ell_k}$ with $I = \{\ell_1, \ell_2, \cdots, \ell_k\}$. The degree of the monomial ξ'_I is $D - |I|$.

By Lemma 6.2 in [4], there exists a set of indices $I \subseteq [D]$, $|I| < s$, such that $g_I \neq 0$ implying $g^\star \neq 0$. Replacing ξ'_I with $\xi_I = \xi_1^{\ell_1}.\xi_2^{\ell_2-\ell_1}\cdots\xi_{k+1}^{D-\ell_k}$ in $g^\star$ retains the non-zeroness of $g^\star$ while the degree of ξ_I becomes D.

▶ Remark 14.

1. Without loss of generality, we assume that the automaton nondeterministically guesses exactly $(s-1)$ indices, i.e., $|I| = s-1$ and the rest as commutative. If $|I| < s-1$, adding more indices still preserves non-zeroness.
2. If the degree of the polynomial g is smaller than $(s-1)$, we will handle this small-degree case separately. For now, we assume $D_1 \geq s-1$ in Lemma 15.

We have the following lemma that shows $\hat{f} \neq 0$.

▶ **Lemma 15.** *Let $f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{ij}$ be a n.c. polynomial over $X = \{x_1, \cdots, x_n\}$, computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Assume that each polynomial Q_{ij} (for $i \in [s], j \in [D_2]$) has degree $D_1 \geq s-1$. Define $\hat{f} \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ as $\hat{f} = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij}$, where $\hat{Q}_{ij} = \sum_{I\subseteq[D_1],|I|=s-1} Q_{i,j,I} \times \xi_I$ and $Q_{i,j,I}$ denotes the polynomial obtained from $Q_{i,j}$ by treating linear forms indexed by I as non-commuting and the remaining linear forms as commuting.*

We will need a generalization of Lemma 15 for polynomials computed by larger depth +-regular circuits.

The resulting n.c. polynomial $\hat{f}$ (from Lemma 15) still has an exponential degree in ξ variables, but each $\hat{Q}_{ij}$ is structured as s-ordered polynomials. Importantly, $\hat{f}$ does not contain any monomials from the spurious polynomial $F = \sum_{m\in Mon(f)} F_m$.

▶ Remark 16. If the degree D_1 of each polynomial Q_{ij} is less than $s-1$ (we call it as small degree case), this case requires a separate treatment.

We now show that each Q_{ij} polynomial can be converted into a structured polynomial $\hat{Q}_{ij}$. The key property of this transformation is that there is a bijection between the monomials of Q_{ij} and those of $\hat{Q}_{ij}$, which ensures that $\hat{Q}_{ij} \equiv 0$ if and only if $Q_{ij} \equiv 0$.

▶ **Proposition 17.** *For each $i \in [s]$, $j \in [D_2]$, let $Q_{ij} \in \mathbb{F}\langle X\rangle$ be a n.c. polynomial computed by $\Sigma\Pi^*\Sigma$ circuit. There exists an explicit substitution automaton of size $O(s)$ that transforms Q_{ij} into a polynomial $\hat{Q}_{ij} \in \mathbb{F}[Y \cup Z]\langle\xi\rangle$ such that $\hat{Q}_{ij} \equiv 0$ if and only if $Q_{ij} \equiv 0$.*

It is easy to observe the following because the first index of each variable $z_{\ell k}$ indicates the position of the variable x_k within each Q_{ij}.

▶ **Observation 18.** *Suppose $\hat{Q}_{ij} \neq 0$. If we treat the variables $z_{\ell k}, \ell \in [c], k \in [n]$, appearing in $\hat{Q}_{ij}$ as commuting, the resulting commutative polynomial $\hat{Q}_{ij}^{(c)}$ remains non-zero.*

This guarantees that for the small degree case, we can transform the polynomial f similarly to Lemma 15, ensuring that each Q_{ij} is transformed into $\hat{Q}_{ij}$ which can be regarded as a commutative polynomial without making it zero. While $\hat{Q}_{ij}$ remains a n.c. polynomial over Z, we acknowledge that our model is black-box and we do not know the value of D_1. However, for the purpose of analyzing the existence of matrices of small dimensions for identity testing, we can assume D_1 is known.

Thus, we can successfully transform the given polynomial in both scenarios – whether $D_1 \geq s-1$ or $D_1 < s-1$ – ensuring that the resulting $\hat{Q}_{ij}$ can be considered as a commutative polynomial without making it a zero polynomial.

However, it is important to note that this transformation alone will not provide a black-box PIT, as we cannot guarantee the non-zeroness of the sum of products of these $\hat{Q}_{ij}$ polynomials. This is because if we simply treat all $\hat{Q}_{ij}$ as commutative, the variables across different $\hat{Q}_{ij}$ polynomials could mix, which may lead to cancellations. At this stage, the variables in Z are still considered n.c. in the polynomial $\hat{f}$.

2.1.2 Non-zeroness of f'

By Lemma 15, we established that $\hat{f} \not\equiv 0$. Next, we show that the polynomial $f' = \hat{f} + F \not\equiv 0$. In $\hat{f}$, for every monomial $m = m_1 m_2 \cdots m_{D_2}$, and for all $\ell \in [D_2]$ each m_ℓ takes the form $\xi_1^{\ell_1}.\xi_2^{\ell_2} \cdots \xi_s^{\ell_s}$ where $\sum_{k\in[s]} \ell_k = D_1$ (see Claim 10). However, this property does not hold for monomials appearing in F (see Claim 11). Specifically, for any monomial $m' = m'_1 m'_2 \cdots m'_N$ in F, there exists a sub-monomial $m'_a = \xi_1^{a_1}.\xi_2^{a_2} \cdots \xi_s^{a_s}$ such that $\sum_{h\in[s]} a_h \neq D_1$. This

distinction ensures that the monomials of $\hat{f}$ do not cancel with those of F. Thus, we conclude that $f' = \hat{f} + F \not\equiv 0$. It's important to note that if $f \equiv 0$ then clearly $f' \equiv 0$ as well (converse statement). We note these observations in the following claim.

▷ Claim 19. Let f be a homogeneous n.c. polynomial computed by a depth-5 +-regular circuit of size s. Then, $f \not\equiv 0$ if and only if $f' = \hat{f} + F \not\equiv 0$.

Next, we can simplify the polynomial f' using the Polynomial Identity Lemma for commutative polynomials. We replace the commuting variables $Y \sqcup Z$ with scalar substitutions from $\mathbb{F}$ or an extension field, yielding a non-zero polynomial. Let us denote this resulting non-zero polynomial as $\tilde{f}$. After this substitution, the only remaining variables in $\tilde{f}$ will be n.c. variables ξ.

Let us denote new polynomials obtained after replacing the commuting variables by scalars in $\hat{f}$ and F by $\hat{f}_1$ and F_1 respectively. That is, $\tilde{f} = \hat{f}_1 + F_1$.

One of the goals of this transformation is to ensure that if ξ_s is followed by ξ_1 in the transformed monomials (for all such occurrences of ξ_s followed by ξ_1), there must be a transition from $Q_{i,j}$ to $Q_{i,j+1}$ for some $j \in [D_2]$. As noted, we cannot be sure of this. However, all those monomials where this transition occurs are captured in the structured part $\hat{f}_1$. Since there is no such structure in F_1, we cannot conclude anything about the monomials appearing in the spurious part F_1.

▶ Remark 20. It is important to note that each monomial of $\tilde{f}$ is a product of ξ-patterns (see Definition 7), and the boundaries of each ξ-pattern can be easily identified by an automaton which is crucially used by the remaining steps.

2.2 Step 2: Product Sparsification

In the second step of our transformation, we prove a *g*eneral lemma that states if we have a sum of a small number of products of ordered polynomials, we can sparsify the product while preserving its non-zero property. Specifically, in each product term of the sum, we can treat only a small number of the ordered polynomials as non-commutative while treating the rest as commutative without affecting the non-zero nature of the polynomial. In particular, this step does not depend on the number of terms in each product.

We focus on the sparsification of the n.c. polynomial $\tilde{f} \in \mathbb{F}\langle\xi\rangle$, which was the output of Step (1). This transformation affects both the good part $\hat{f}_1$ and the spurious part F_1 of the polynomial $\tilde{f} \in \mathbb{F}\langle\xi\rangle$. We begin by analyzing the transformation of $\hat{f}_1$, which is defined as:

$$\hat{f}_1 = \sum_{i\in[s]} \left(\prod_{j\in[D_2]} \hat{Q}_{ij} \right),$$

where each $\hat{Q}_{ij}$ is an s-ordered polynomial in the n.c. variables $\xi = \{\xi_1, \ldots, \xi_s\}$. Note that each $\hat{Q}_{ij}$ is a homogeneous and degree D_1 n.c. polynomial.

The key observation is that we can preserve the non-zeroness of $\hat{f}_1$ by retaining at most $s-1$ of the s-ordered polynomials $\hat{Q}_{ij}$ in each product $\prod_{j\in[D_2]} \hat{Q}_{ij}$ as non-commutative while treating the remaining ones as commutative. This is stated in the following lemma, which we refer to as the *product sparsification lemma*. This lemma generalizes Lemma 6.2 from [4]. However, unlike in [4], we are working with the product of non-commutative polynomials where the degree of individual factors can be greater than 1. If we simply treat them as commutative, as in [4], they may become zero. The proof of this lemma crucially relies on Claim 5. Unlike [4], one of the key distinctions in our setting is that the $\hat{Q}_{ij}$ polynomial can be non-homogeneous in general.

The product sparsification step impacts both $\hat{f}_1$ and the spurious part F_1 of the polynomial $\tilde{f}$ obtained after Step 1. We will address the effects on both components in a later step (see 2.4).

▶ **Lemma 21** (Product Sparsification Lemma). *Let $\hat{f}_1 = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij}$, where each $\hat{Q}_{ij}$ is an s-ordered polynomial of degree D_1 over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$. Then, there exists a subset $I \subseteq [D_2]$ with size at most $s-1$ such that if we treat the polynomials $\hat{Q}_{ij}$ for $j \in I$, as non-commutative and the others ($j \notin I$) as commutative, then the polynomial $\hat{f}_1$ remains non-zero. Furthermore, each $\hat{Q}_{ij}$ polynomial may be non-homogeneous in general. Moreover, there is a small substitution automaton of size $O(s)$ that performs this transformation.*

In other words, for each n.c. variable ξ_i (where $i \in [s]$) in $\hat{f}_1$, there exists an $O(s)$-dimensional matrix – acting as a transition matrix of a substitution automaton of size $O(s)$. By evaluating $\hat{f}_1$ on these matrices, the polynomial is transformed into a product-sparsified polynomial while maintaining its non-zero property.

▶ Remark 22. We remark that the proof of Lemma 21 relies solely on the fact that each $\hat{Q}_{ij}$ polynomial in $\hat{f}_1$ is an ordered polynomial. In particular, the proof does not depend on the fact that each $\hat{Q}_{ij}$ is obtained from a $\Sigma\Pi^*\Sigma$ circuit. Instead, the proof relies on the following facts:

1. the number of summands is small,
2. the boundary of each ordered polynomial can be efficiently identified using a small automaton

This makes it irrelevant where the $\hat{Q}_{ij}$ polynomials originate from. As a result, we can apply this result whenever the given polynomial is represented as a sum of a small number of products of ordered polynomials (i.e., the number of summands is small). We will use this observation when working with higher-depth +-regular circuits.

Since we do not know the index set I, the substitution automaton guesses the index set I. Since the index set $I \subseteq [D_2]$ is unknown, the automaton non-deterministically selects which $\hat{Q}_{ij}$ polynomials will be treated as non-commutative. Given the structured nature of the polynomial $\hat{f}$, we can identify the *boundary* of each $\hat{Q}_{ij}$, ensuring that no additional spurious monomials are generated.

In the high-degree case ($D_1 \geq s-1$), either ξ_s or ξ_{s-1} followed by ξ_1 indicates the end of each $\hat{Q}_{ij}$, which can be easily recognized by the automaton. In the low-degree case ($D_1 < s-1$), the smaller degree allows us to identify the ends of each $\hat{Q}_{ij}$ with a small automaton of size at most $s-2$.

The substitution automaton selects at most $(s-1)$ of the $\hat{Q}_{ij}$ polynomials to be treated as n.c. while treating the remaining ones as commutative. The ξ variables in the chosen commutative polynomials $\hat{Q}_{ij}$ are substituted with fresh commutative variables $\zeta = \{\zeta_1, \ldots, \zeta_s\}$. In the selected commutative polynomials $\hat{Q}_{ij}$, each n.c. variable $\xi_k, k \in [s]$ is replaced by the corresponding commuting variable ζ_k. Additionally, to distinguish between different guesses made by the substitution automaton, we use fresh commutative block variables $\chi = \{\chi_1, \ldots, \chi_s\}$.

Let $J = \{j_1, j_2, \ldots, j_{s-1}\} \subseteq [D_2]$ with $j_1 < j_2 < \ldots < j_{s-1}$.
We define $\chi_J = \chi_1^{j_1-1} \cdot \chi_2^{j_2-j_1-1} \ldots \chi_s^{D_2-j_{s-1}}$. If the automaton guesses the $\hat{Q}_{ij}$ polynomials corresponding to the positions in the index set J as n.c., the output g_J of the substitution automaton for this specific guess J will be

$$g_J = \sum_{i\in[s]} \left(\prod_{j\in\overline{J}} \hat{Q}_{ij}\right)\left(\prod_{j\in J} \hat{Q}_{ij}\right) \times \chi_J. \tag{7}$$

Note that $\left(\prod_{j\in\overline{J}} \hat{Q}_{ij}\right)$ is a commutative polynomial over $\zeta = \{\zeta_1, \ldots, \zeta_s\}$. We have the following proposition about the output of the substitution automaton, whose proof we omit as it follows by a straightforward argument.

▶ **Proposition 23.** *Let $\hat{f}_1 \in \mathbb{F}\langle\xi\rangle$ be the structured part of the polynomial obtained after Step 1. Let $\hat{f}_1'$ be the output of the substitution automaton on the structured polynomial $\hat{f}_1$ and it can be expressed as*

$$\hat{f}_1' = \sum_{J\subseteq[D_2],|J|=s-1} g_J.$$

Moreover, $\hat{f}_1 \neq 0$ if and only if $\hat{f}_1' \neq 0$.

It's evident that for distinct guesses J and J' where $J \neq J'$, the monomials of g_J and g'_J will not mix, since the sub-monomials χ_J and $\chi_{J'}$ are distinct (see Equation 7). By Lemma 21, there exists index set $J \subseteq [D_2]$ with size at most $s-1$, such that $g_J \neq 0$ implying $\hat{f}_1' \neq 0$.

Next, we can simplify $\hat{f}_1'$ by using the Polynomial Identity Lemma for commutative polynomials to eliminate the commuting variables $\zeta \cup \chi$ by substituting scalars. As a result, the remaining variables in the polynomial will be solely the n.c. variables ξ.

Let us denote the new polynomial obtained after replacing the commuting variables by scalars in $\hat{f}_1{}'$ by $\hat{f}_2$.

This product sparsification step affects both the good part $\hat{f}_1$ and the spurious part F_1 of the polynomial $\tilde{f}$ obtained after Step 1. We will denote the new polynomial derived from the spurious part F_1 by F_2. In Step 2, we apply product sparsification to both $\hat{f}_1$ and F_1, which yields the n.c. polynomials $\hat{f}_2$ and F_2 respectively (with all commuting variables replaced by scalars).

2.3 Step 3: Commutative Transformation of $\hat{f}_2$

In this final step, we prove a general commutative transformation lemma, which states that if we have a non-commutative polynomial represented as a sum of products of a small number of ordered polynomials (i.e., the number of terms in each product is small), we can convert it into a commutative polynomial while preserving its non-zeroness property. In particular, this step does not depend on the number of summands. The key idea is to introduce a small number of new commutative variables to perform this transformation.

We now describe how to transform $\hat{f}_2$ into a commutative polynomial while preserving its non-zeroness. Note that $\hat{f}_2$ is a polynomial over $\mathbb{F}\langle\xi\rangle$. If we treat $\hat{f}_2$ as commutative by considering the n.c. variables ξ as commutative, the exponents of the variable ξ_i (for $i \in [s]$) from different n.c. $\hat{Q}_{ij}$ polynomials will be summed (or mixed). This mixing makes it impossible to guarantee that the resulting polynomial remains non-zero.

However, we can carefully convert $\hat{f}_2$ into a commutative polynomial while preserving its non-zeroness. This is stated in the following lemma. In particular, there is a substitution automaton of size $O(s^2)$ that carries out this commutative transformation.

▶ **Lemma 24** (Commutative Transformation Lemma). *Let $g = \sum_{i\in[s]} \beta_i \left(\prod_{j\in[s]} \hat{Q}_{ij}\right)$, where $\beta_i \in \mathbb{F}$ and each $\hat{Q}_{ij}$ is an s-ordered polynomial over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ of degree D. This can be expressed as $g = \sum_m \alpha_m m$, where each monomial m has the form $m = \prod_{j\in[s]} \xi_1^{i_{j1}} \xi_2^{i_{j2}} \ldots \xi_s^{i_{js}}$. Then there exists a substitution automaton of size $O(s^2)$ that transforms the non-commutative polynomial g into a commutative polynomial $g^{(c)}$ while preserving non-zeroness. In particular, $g^{(c)} \equiv 0 \iff g \equiv 0$.*

In other words, for each n.c. variable ξ_i (where $i \in [s]$), there exists an $O(s^2)$-dimensional matrix – acting as a transition matrix of a substitution automaton. By evaluating g on these matrices, the polynomial is transformed into a commutative polynomial $g^{(c)}$, while maintaining its non-zero property.

▶ Remark 25. We remark that Lemma 24 is more general. The proof depends only on the fact that the given non-commutative polynomial can be represented as a sum of products of a small number of ordered polynomials (i.e., the number of terms in each product is small). Crucially, the proof of this commutative transformation does not depend on the fact that the polynomials are derived from +-regular circuits or whether they are homogeneous. Instead, the proof relies on the following two facts:

1. The number of terms in each product is small, and
2. Each term in the product is an ordered polynomial so that boundaries can be identified efficiently using an automaton.

This makes the result applicable whenever the given non-commutative polynomial is represented as a sum of products of a small number of ordered polynomials. We will use this observation when working with higher-depth +-regular circuits.

By applying Lemma 24, we can transform the polynomial $\hat{f}_2$, the structured part obtained after Step 2, into a commutative polynomial while preserving its non-zeroness. Let $\hat{f}_3^{(c)}$ denote the resulting commutative polynomial derived from $\hat{f}_2$. Consequently, we establish that $\hat{f}_3^{(c)} \not\equiv 0$ as a result of this lemma.

Next, given $\tilde{f} = \hat{f}_1 + F_1$, where $\tilde{f}$ was obtained after Step 1, we can likewise transform $\tilde{f}$ into a commutative polynomial. Let $F_3^{(c)}$ represent the commutative polynomial obtained from F after applying steps (2) and (3). If $\hat{f}_3^{(c)} + F_3^{(c)} \not\equiv 0$, we have successfully converted a n.c. polynomial f, computed by a depth-5 +-regular circuit, into a commutative polynomial that preserves non-zeroness. We can now check the non-zeroness of this commutative polynomial using the Polynomial Identity Lemma for commutative polynomials.

Assume $\hat{f}_3^{(c)} + F_3^{(c)} = 0$. We will now detail how to modify the coefficients of certain monomials in $\tilde{f}$, which was obtained in Step 1, before executing Steps (2) and (3). We establish that this coefficient modification maintains non-zeroness and remains non-zero even after the application of Steps (2) and (3).

2.4 Coefficient Modification by Modulo Counting Automaton

Assuming $\hat{f}_3^{(c)} + F_3^{(c)} = 0$, and given that $\hat{f}_3^{(c)} \not\equiv 0$, it follows that $F_3^{(c)} \not\equiv 0$ and $\hat{f}_3^{(c)} = -F_3^{(c)}$. To address this cancellation, we carefully modify certain monomial coefficients in the non-commutative polynomial $\tilde{f} = \hat{f}_1 + F_1$ prior to applying product sparsification (Lemma 21) and commutative transformation (Lemma 24). We show that these modifications ensure that the resulting polynomial remains non-zero after Steps (2) and (3).

2.5 Black-box Randomized PIT for $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ Circuits

Each of these three steps, along with the coefficient modification step, results in its own set of matrices for evaluation. In particular, the matrices obtained in each step evaluate a n.c. polynomial derived from the previous step.

Given that our model operates as a black box, we cannot evaluate the polynomial in this manner. Instead, we require a single matrix substitution for each n.c. variable. To address this, we apply the matrix composition lemma to combine the substitution matrices from all

four steps into a single matrix for each n.c. variable. This approach allows us to establish an efficient randomized polynomial identity testing (PIT) algorithm for depth-5 +-regular circuits, as demonstrated in the following theorem.

▶ **Theorem 26.** *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$, computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then $f \not\equiv 0$ if and only if it does not evaluate to zero on the matrix algebra $\mathbb{M}_{s^6}(\mathbb{F})$.*

The main idea of the proof is to compose the substitution matrices derived from the automata in Steps (1)–(3) using the matrix composition lemma.

▶ Remark 27. We observe that the commutative polynomial $f^{(c)} \in \mathbb{F}[W]$ is an s^2-ordered polynomial over W, in the sense that we can arrange the variables in each monomial of $f^{(c)}$ in increasing order according to the first index of the W variables, allowing for some exponents to be zero as specified in Definition 3.

This is summarized in the following theorem.

▶ **Theorem 28.** *Let f be a non-zero non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then, f can be transformed into an s^2-ordered polynomial while preserving its non-zeroness. In particular, there exists a small substitution automaton of size $O(s^6)$ that performs this transformation.*

In other words, for each n.c. input variable x_i (where $i \in [n]$) in f, there exists an $O(s^6)$-dimensional matrix – acting as a transition matrix of a substitution automaton of size $O(s^6)$. By evaluating f on these matrices, the polynomial f is transformed into an s^2-ordered polynomial while preserving its non-zeroness (in one particular entry of the resulting matrix).

2.5.1 An Automaton for Theorem 28

We can envision a substitution automaton $\mathcal{A}$ for Theorem 28 as follows. By applying the matrix composition Lemma, we can combine the substitution matrices obtained from Steps (1) through (3), along with the modifications to coefficients, into a single substitution matrix $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ of dimension $O(s^6)$. We then evaluate the polynomial as $\mathbf{O} = f(\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$.

The output of the substitution automaton is defined as the sum of several entries of the matrix $\mathbf{O}$ (refer to the polynomial $f^{(c)}$ defined in the proof of Theorem 26).

It is crucial to note that each entry of the matrices in $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ is a monomial over $Y \sqcup Z \sqcup \zeta \sqcup \chi \sqcup W$, where $Y \sqcup Z$ are commutative variables from Step (1), and $\zeta \sqcup \chi$ are commutative variables from Step (2), while W contains commutative variables from Step (3).

As noted in Steps (1) and (2), we can replace the commutative variables in $Y \sqcup Z \sqcup \zeta \sqcup \chi$ with scalars without losing the non-zeroness of the output polynomial. Since these commutative variables are disjoint, for simplicity in the analysis, we substitute them with scalars. That is, there exist scalar substitutions for the variables in $Y \sqcup Z \sqcup \zeta \sqcup \chi$ such that non-zeroness is preserved. To avoid carrying these variables throughout all derivations, we replace them with scalars, and by the DeMillo–Lipton–Schwartz–Zippel lemma, such substitutions are guaranteed to exist. After these replacements, each entry of the matrices in $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ transforms into scalar multiples of variables over W. We denote the resulting matrices as $\mathbf{M'} = (\mathbf{M'_{x_1}}, \mathbf{M'_{x_2}}, \ldots, \mathbf{M'_{x_n}})$.

We can construct a substitution automaton $\mathcal{A}$ such that the substitution matrix for the variable x_i is given by the matrix $\mathbf{M}'_{\mathbf{x_i}}$, where the entries are scalar multiples of variables in W. These entries correspond to transitions that substitute a n.c. variable with a scalar multiple of a variable in W. This automaton $\mathcal{A}$ effectively transforms f into an s^2-ordered polynomial $f^{(c)}$ while preserving its non-zeroness.

We can view the resulting s^2-ordered polynomial $f^{(c)}$ as a n.c. polynomial over W. This idea is crucial for developing black-box randomized polynomial identity testing (PIT) for circuits of larger depths using induction.

▶ Remark 29. Note that we only *view* $f^{(c)}$ as a n.c. polynomial and no explicit conversion or transformation is done.

It is important to note that the monomials of $f^{(c)}$ do not correspond to a single entry of the output matrix $\mathbf{O} = f(\mathbf{M}'_{\mathbf{x_1}}, \mathbf{M}'_{\mathbf{x_2}}, \ldots, \mathbf{M}'_{\mathbf{x_n}})$. Instead, they represent the sum of several entries, as indicated in the polynomial $f^{(c)}$ defined in the proof of Theorem 26. Effectively, the column numbers of these entries form the set of accepting states for the new automaton $\mathcal{A}$, with row 1 serving as the starting state of this automaton.

3 Black-Box Randomized PIT for Small Depth +-Regular Circuits

In this section, we present an efficient randomized black-box polynomial identity testing (PIT) algorithm for polynomials computed by small-depth +-regular circuits.

▶ **Theorem 30.** *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$, computed by a +-regular circuit of size s and depth d. Then, $f \not\equiv 0$ if and only if f is not identically zero on $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and $|\mathbb{F}|$ is sufficiently large.*

3.1 Transforming f into an ordered polynomial

Similar to depth-5 case, the polynomial f computed by a size s depth d +-regular circuit can be converted into an ordered polynomial using a substitution automaton of size at most $s^{O(d^2)}$. We state the result in the following theorem.

▶ **Theorem 31.** *Let f be a non-zero n.c. polynomial of degree D over variables $X = \{x_1, \ldots, x_n\}$, computed by a +-regular circuit of size s and depth d. Let d^+ denote the number of Σ layers in the circuit. Then there exists a substitution automaton $\mathcal{A}$ of size at most $s^{O(d^2)}$ such that the polynomial $\hat{f} := f(\mathcal{A})$ is an $s^{(d^+-1)}$-ordered polynomial. Moreover, this transformation preserves non-zeroness: $\hat{f} \equiv 0 \iff f \equiv 0$.*

3.2 Randomized Identity Test for Small Depth +-Regular Circuits

We are now ready to state and prove the main theorem.

▶ **Theorem 32.** *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$, computed by a +-regular circuit of size s and depth d. We denote the number of addition (i.e., $\sum$) layers in the circuit by d^+. Then, $f \not\equiv 0$ if and only if f is not identically zero on $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and $|\mathbb{F}|$ is sufficiently large.*

Proof. We use Theorem 31 to convert the n.c. polynomial f into an $s^{(d^+-1)}$-ordered polynomial f_{ops}, while preserving its non-zeroness. As discussed above, there is a substitution automaton of size bounded by $s^{O(d^2)}$, which results in substitution matrices of dimension $s^{O(d^2)}$. By Claim 5, f_{ops} can be treated as a commutative polynomial while preserving its

non-zeroness. Using the DeMillo-Lipton-Schwartz-Zippel lemma, we can have a randomized PIT for depth d +-regular circuit of size s using matrices of dimension at most $s^{O(d^2)}$. This completes the proof of the theorem. ◀

References

1 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. doi:10.1137/140975103.

2 Avraham Shimshon Amitsur and Jacob Levitzki. Minimal identities for algebras. *Proceedings of the American Mathematical Society*, 1(4):449–463, 1950.

3 Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial time identity testing for noncommutative circuits. 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 831–841. ACM, 2017. doi:10.1145/3055399.3055442.

4 Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial-time identity testing for noncommutative circuits. *Theory of Computing*, 15:1–36, 2019. doi:10.4086/toc.2019.v015a007.

5 G. V. Sumukha Bharadwaj and S. Raja. Randomized black-box PIT for small depth +-regular non-commutative circuits, 2025. doi:10.48550/arXiv.2411.06569.

6 Andrej Bogdanov and Hoeteck Wee. More on noncommutative polynomial identity testing. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 92–99. IEEE, 2005. doi:10.1109/CCC.2005.13.

7 Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 243–252. IEEE, 2013. doi:10.1109/FOCS.2013.34.

8 Laurent Hyafil. The power of commutativity. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 171–174. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.31.

9 Oscar H. Ibarra and Shlomo Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the ACM (JACM)*, 30(1):217–228, 1983. doi:10.1145/322358.322373.

10 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418, 1991.

11 Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005. doi:10.1007/s00037-005-0188-8.

Unreliability in Practical Subclasses of Communicating Systems

Amrita Suresh
University of Oxford, UK

Nobuko Yoshida
University of Oxford, UK

Abstract

Systems of communicating automata are prominent models for peer-to-peer message-passing over unbounded channels, but in the general scenario, most verification properties are undecidable. To address this issue, two decidable subclasses, *Realisable with Synchronous Communication* (RSC) and *k-Multiparty Compatibility* (*k*-MC), were proposed in the literature, with corresponding verification tools developed and applied in practice. Unfortunately, both RSC and *k*-MC are not resilient under failures: (1) their decidability relies on the assumption of perfect channels and (2) most standard protocols do not satisfy RSC or *k*-MC under failures. To address these limitations, this paper studies the resilience of RSC and *k*-MC under two distinct failure models: *interference* and *crash-stop failures*. For interference, we relax the conditions of RSC and *k*-MC and prove that the inclusions of these relaxed properties remain decidable under interference, preserving their known complexity bounds. We then propose a novel crash-handling communicating system that captures wider behaviours than existing multiparty session types (MPST) with crash-stop failures. We study a translation of MPST with crash-stop failures into this system integrating RSC and *k*-MC properties, and establish their decidability results. Finally, by verifying representative protocols from the literature using RSC and *k*-MC tools extended to interferences, we evaluate the relaxed systems and demonstrate their resilience.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Distributed computing models; Theory of computation → Automata extensions

Keywords and phrases Communicating automata, lossy channel, corruption, out of order, session types, crash-stop failure

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.52

Related Version *Full Version*: `http://arxiv.org/abs/2510.03941`

Supplementary Material *Software*: `https://github.com/NobukoYoshida/Interference-Tool`

Funding This work is supported by EPSRC EP/T006544/2, EP/T014709/2, EP/Y005244/1, EP/V000462/1, EP/X015955/1, EP/Z0005801/1, EU Horizon TARDIS 101093006 (UKRI number 10066667) and ARIA.

Acknowledgements We thank Martin Vassor for his comments on an early version of this paper.

1 Introduction

Asynchronous processes that communicate using First In First Out (FIFO) channels [11], henceforth referred to as FIFO systems, are widely used to model distributed protocols, but their verification is known to be computationally challenging. The model is Turing-powerful for even just two processes communicating via two unidirectional FIFO channels [11].

To address this challenge, several efforts have focused on identifying practical yet decidable subclasses – those expressive enough to model a wide range of distributed protocols, while ensuring that verification problems such as reachability and model checking remain decidable. Most FIFO systems assume *perfect* channels, which is too restrictive to model the real-world


© Amrita Suresh and Nobuko Yoshida;
licensed under Creative Commons License CC-BY 4.0

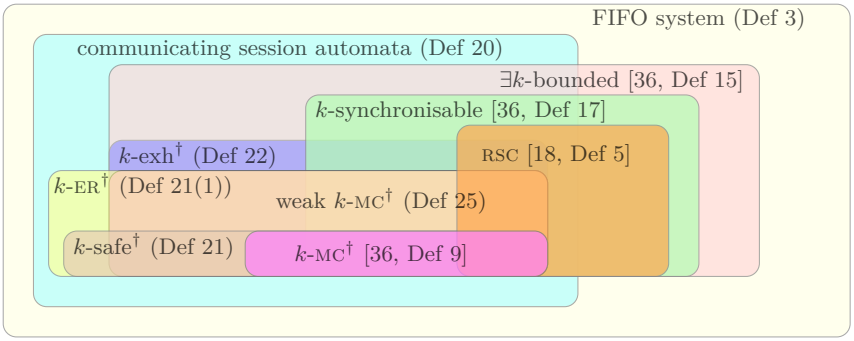


Figure 1 Classes of communication systems (since the †-marked definitions are introduced in the context of CSA (Def 20), we restrict them accordingly).

distributed phenomena where system failures often happen. This paper investigates whether two practical decidable subclasses of communicating systems, *Realisable with Synchronous Communication* (RSC) [18] and *k-Multiparty Compatibility* (*k*-MC) [35], are *resilient* when integrated with two different kinds of *failures*. These failure models were originally introduced in the contexts of *contracts* [37] and *session types* [3, 6]. We say a system is *resilient* under a given failure model if (i) the inclusion remains decidable, and (ii) the verification properties of interest remain decidable under that failure model.

Failures in communications. A widely studied failure model in FIFO systems is *lossy channels*. Finkel [21] showed that the termination problem is decidable for the class of *completely specified protocols*, a model which strictly includes FIFO systems with lossy channels. Abdulla and Jonsson [1] developed algorithms for verifying termination, safety, and eventuality properties for protocols on lossy channels, by showing that they belong to the class of well-structured transition systems.

Another type of failure, studied in a more practical setting, occurs when one or more processes *crash*. In the most general case, Fekete et al. [20] proved that if an underlying process crashes, no fault-tolerant reliable communication protocol can be implemented. To address this, they consider faultless models which attempt to capture the behaviour of crashes by broadcasting crash messages. Such approaches have been explored in the context of *runtime verification* techniques [31] and *session types* [3, 5, 6]. In this work, we closely study a failure model proposed by [3, 5].

Restricting the channel behaviour. To define decidable subclasses, many works study how to restrict read and write access to channels. For two-process (binary) FIFO systems, the notion of *half-duplex* communication was introduced in [14], where at most one direction of communication is active at any time. For such systems, reachability is decidable in polynomial time. However, generalising this idea to the multiparty setting often yields subclasses that are either too restrictive or lose decidability.

Di Giusto et al. [17, 18] extended this idea to multiparty systems while preserving decidability, resulting in the notion of systems *realisable with synchronous communication* (RSC). They showed that this definition overlaps with that in [14] for mailbox communication. However, in the case of peer-to-peer communication where the two definitions differ, peer-to-peer RSC behaviour was proved to be decidable. RSC systems are related to *synchronisable systems* [8, 10, 19], in which FIFO behaviours must admit a synchronisable execution. The tool *ReSCu* applies this idea to verify real-world distributed protocols [16].

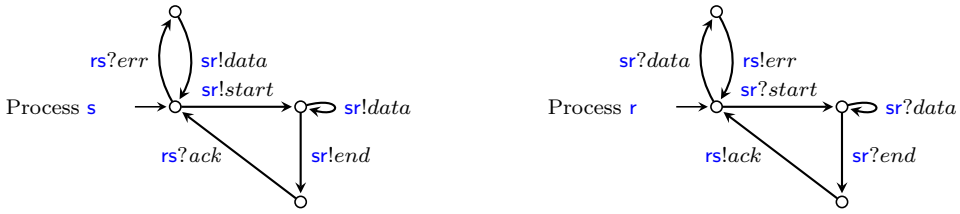


Figure 2 The above system $\mathcal{S}$ is half-duplex in the absence of errors. However, in case of (any or multiple) errors, it is no longer half-duplex.

Another approach to restricting channel behaviours is to bound the length of the channel. Lohrey [36] introduced *existentially bounded systems* (see also [23, 24]) where all executions that reach a final state with empty channels can be re-ordered into a bounded execution. Although many verification problems are decidable for this class of systems, checking if a system is existentially k-bounded is undecidable, even if k is given as part of the input.

A decidable bounded approach, *k-multiparty compatibility* (k-MC), was introduced in [35]. This property is defined by two conditions, *exhaustivity* and *safety*. Exhaustivity implies existential boundedness and characterises systems where each automaton behaves the same way under bounds of a certain length. Checking k-MC is decidable, and the tool k-MC-checker is implemented and applied to verify Rust [12, 32] and OCaml [30] programs.

Combining the two approaches. As far as we are aware, the intersection between expressive, decidable subclasses and communication failures is less explored. Lozes and Villard [37] studied reliability in binary half-duplex systems and showed that many communicating contracts can be verified with this model. Inspired by this, we investigate whether practical *multiparty* subclasses, RSC and k-MC, remain robust in the presence of communication errors.

Although failure models such as lossy channels are well studied, the complexity of verification in their presence is often very high – for instance, reachability in lossy systems is non-primitive recursive [21]. Our goal is not only to show that RSC and k-MC systems are resilient, but also that their inclusion remains decidable under failure models, with complexity maintained from the failure-free case.

This paper extends RSC [17, 18] and k-MC [35] by integrating two distinct failure models. RSC and k-MC systems are incomparable to each other (RSC is not a subset of k-MC and vice-versa), but both are closely related to existentially bounded systems. Figure 1 illustrates their relationship with other models.

For failures, first, we consider *interferences* from the environments by modelling *lossiness*, *corruption* (a message is altered to a different message) and *out-of-ordering* (two messages in a queue are swapped) of channels studied in the context of FIFO systems. Secondly, we consider potential *crashes* of processes introduced in the setting of session types [3, 6].

Let us consider the following simple half-duplex protocol as an example.

▶ **Example 1.** The system in Figure 2 is half-duplex under the assumption of perfect channels [14]. It consists of two processes, a sender (s) and a (dual) receiver (r), communicating via unbounded FIFO channels. A transition sr!m denotes that the sender puts (asynchronously) a message m on channel sr, and similarly, sr?m denotes that message m is consumed by the receiver from channel sr. Since the channel rs only contains messages after the receiver receives the *end* message and has emptied sr, the system satisfies the half-duplex condition. Moreover, the sender never sends *data* without having first sent *start* so the error loop is never triggered.

Now suppose the channels are prone to *corruption*. A message *data* could be altered to *end* after being sent. This allows the receiver to react prematurely by sending *ack*, while the sender continues sending *data*. As a result, both channels may become non-empty, violating the half-duplex property. Similarly in the presence of other forms of interference, as shown in Example 6, this system no longer satisfies the half-duplex condition.

Contributions and outline. The main objective of this paper is to investigate whether multiparty adaptations of half-duplex systems (RSC and k-MC) retain both their expressiveness for modelling real-world protocols and the decidability of their inclusions, with preserved complexity, under two distinct kinds of communication failures: interferences and crash-stops. § 2 introduces preliminary notions, notably FIFO systems and interference models; § 3 studies RSC under interference, and shows that relaxing certain conditions on matching send and receive actions retains both expressiveness and decidability (Theorem 19). § 4 examines k-MC with interferences, and proposes a relaxed version, k-WMC (weak k-MC), by weakening the *safety* condition. We prove that checking the k-WMC property remains decidable under interferences (Theorem 26). § 5 introduces the FIFO systems with crash-stop failures (called *crash-handling systems*), and shows that checking RSC and k-WMC under crash-stop failures is decidable (Theorems 32 and 33); § 6 defines a translation from (local) multiparty session types (MPST) to crash-handling systems and proves that this translation preserves trace semantics. This implies the decidability of RSC and k-MC within the asynchronous MPST system extended to crash-stop failures (Theorem 40); § 7 evaluates protocols from the literature extending the existing tools with support for interferences; and § 8 concludes with further related and future work. Proofs are provided in the appendix and [45]. The tools and benchmarks are available from `https://github.com/NobukoYoshida/Interference-Tool`.

2 Preliminaries

For a finite set Σ, we denote by Σ^* the set of finite words over Σ, and the empty word with ε. We use $|w|$ to denote the length of the word w, and $w_1 \cdot w_2$ indicates the concatenation of two words $w_1, w_2 \in \Sigma^*$. Given a (non-deterministic) finite-state automaton $\mathcal{A}$, we denote by $\mathcal{L}(\mathcal{A})$ the language accepted by $\mathcal{A}$. Consider a finite set of processes $\mathbb{P}$ (ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$ or occasionally by $\mathsf{r}_1, \mathsf{r}_2, \ldots$) and a set of messages Σ. In this paper, we consider the *peer-to-peer* communication model; i.e., there is a pair of unidirectional FIFO channels between each pair of processes, one for each direction of communication. In our model, processes act either by point-to-point communication or by internal actions (actions local to a single process). Moreover, in this setting, we consider messages to be atomic, akin to letters of an alphabet.

Let $Ch = \{\mathsf{pq} \mid \mathsf{p} \neq \mathsf{q} \text{ and } \mathsf{p}, \mathsf{q} \in \mathbb{P}\}$ be a set of *channels* that stand for point-to-point links. Since we are considering the peer-to-peer model of communication, there is a unique process that can send a message to (or dually, receive a message from) a particular channel. An action $a = (\mathsf{pq}, !, m) \in \mathsf{Act}$ indicates that a process p sends a message m on the channel pq. Similarly, $a = (\mathsf{qp}, ?, m) \in \mathsf{Act}$ indicates that p receives a message m on the channel qp. We henceforth denote an action $a = (\mathsf{pq}, \dagger, m) \in \mathsf{Act}$, where $\dagger \in \{!, ?\}$, in a simplified form as $\mathsf{pq}\dagger m$. An *internal action* c_p means that process p performs the action c. We define a finite set of actions as $\mathsf{Act} \subseteq (Ch \times \{!, ?\} \times \Sigma) \cup \mathsf{Act}_\tau$ where Act_τ is the set of all internal actions.

▶ **Definition 2** (FIFO automaton). *A FIFO automaton $\mathcal{A}_\mathsf{p}$, associated with p, is defined as $\mathcal{A}_\mathsf{p} = (Q_\mathsf{p}, \delta_\mathsf{p}, q_{0\mathsf{p}})$ where: Q_p is the finite set of control-states, $\delta_\mathsf{p} \subseteq Q_\mathsf{p} \times \mathsf{Act} \times Q_\mathsf{p}$ is the transition relation, and $q_{0\mathsf{p}} \in Q_\mathsf{p}$ is the initial control-state.*

Note that in this model, there are no final or accepting states.

The set of outgoing channels of process p is represented by $Ch_{\mathsf{o},\mathsf{p}} = \{\mathsf{pq} \mid \mathsf{q} \in \mathbb{P} \setminus \mathsf{p}\}$. Similarly, $Ch_{\mathsf{i},\mathsf{p}} = \{\mathsf{qp} \mid \mathsf{q} \in \mathbb{P} \setminus \mathsf{p}\}$ is the set of incoming channels of process p.

Given an action a, an *active process*, denoted by $\mathsf{proc}(a)$, is defined as: $\mathsf{proc}(\mathsf{pq}!m) = \mathsf{p}$ and $\mathsf{proc}(\mathsf{pq}?m) = \mathsf{q}$. Similarly, $\mathsf{ch}(\mathsf{pq}!m) = \mathsf{ch}(\mathsf{pq}?m) = \mathsf{pq}$.

We say a control state $q \in Q_{\mathsf{p}}$ is a *sending state* (resp. *receiving state*) if all its outgoing transitions are labelled by send (resp. receive) actions. If a control state is neither a sending nor receiving state, i.e., it either has both send and receive actions or neither, then we call it a *mixed state*. We say a sending (resp. receiving) state is *directed* if all the send (resp. receive) actions from that control state are to the same process. Like for finite-state automata, we say that a FIFO automaton $\mathcal{A}_{\mathsf{p}} = (Q_{\mathsf{p}}, \delta_{\mathsf{p}}, q_{0\mathsf{p}})$ is *deterministic* if for all transitions $(q, a, q'), (q, a', q'') \in \delta_{\mathsf{p}}$, $a = a' \implies q' = q''$. We write $q_1 \xrightarrow{a_1 \cdots a_n} q_{n+1}$ for $(q_1, a_1, q_2) \cdots (q_n, a_n, q_{n+1}) \in \delta_{\mathsf{p}}$. Unless specified otherwise, we consider non-deterministic automata, allowing mixed states, and all states do not have to be directed.

▶ **Definition 3** (FIFO system). *A FIFO system $\mathcal{S} = (\mathcal{A}_{\mathsf{p}})_{\mathsf{p}\in\mathbb{P}}$ is a set of communicating FIFO automata. A configuration of $\mathcal{S}$ is a pair $\gamma = (\vec{q}; \vec{w})$ where $\vec{q} = (q_{\mathsf{p}})_{\mathsf{p}\in\mathbb{P}}$ is called the* global state *with $q_{\mathsf{p}} \in Q_{\mathsf{p}}$ being one of the local control-states of $\mathcal{A}_{\mathsf{p}}$, and where $\vec{w} = (w_{\mathsf{pq}})_{\mathsf{pq}\in Ch}$ with $w_{\mathsf{pq}} \in \Sigma^*$.*

The *initial configuration* of $\mathcal{S}$ is $\gamma_0 = (\vec{q_0}; \vec{\varepsilon})$ where $\vec{q_0} = (q_{0\mathsf{p}})_{\mathsf{p}\in\mathbb{P}}$ and we write $\vec{\varepsilon}$ for the $|Ch|$-tuple $(\varepsilon, \ldots, \varepsilon)$. We let $\mathcal{A}_{\mathsf{p}} = (Q_{\mathsf{p}}, \delta_{\mathsf{p}}, q_{0\mathsf{p}})$ be a FIFO automaton. Let $\mathcal{S} = (\mathcal{A}_{\mathsf{p}})_{\mathsf{p}\in\mathbb{P}}$ be the system whose initial configuration is γ_0. The FIFO automaton $\mathsf{product}(\mathcal{S})$ associated with $\mathcal{S}$ is the standard asynchronous product automaton: $\mathsf{product}(\mathcal{S}) = (Q, \delta, \vec{q_0})$ where $Q = \prod_{\mathsf{p}\in\mathbb{P}} Q_{\mathsf{p}}$, $\vec{q_0} = (q_{0\mathsf{p}})_{\mathsf{p}\in\mathbb{P}}$, and δ is the set of triples $(\vec{q_1}, a, \vec{q_2})$ for which there exists $\mathsf{p} \in \mathbb{P}$ such that $(q_{1_{\mathsf{p}}}, a, q_{2_{\mathsf{p}}}) \in \delta_{\mathsf{p}}$ and $q_{1_{\mathsf{r}}} = q_{2_{\mathsf{r}}}$ for all $\mathsf{r} \in \mathbb{P} \setminus \{\mathsf{p}\}$. An *execution* $e = a_1 \cdot a_2 \cdots a_n \in \mathsf{Act}^*$ is an arbitrary finite sequence of actions. We write $\mathsf{executions}(\mathcal{S})$ for $\{e \in \mathsf{Act}^* \mid \gamma_0 \xrightarrow{e} \gamma \text{ for some configuration } \gamma\}$. Given $e = a_1 \cdot a_2 \cdots a_n$, we write $\mathsf{Act}(e) = \{a_1, a_2, \ldots, a_n\}$. Moreover, we say two systems are trace-equivalent if they produce the same set of executions, i.e. $\mathcal{S} \approx \mathcal{S}'$ is as follows: $\forall \phi, \phi \in \mathsf{executions}(\mathcal{S}) \Leftrightarrow \phi \in \mathsf{executions}(\mathcal{S}')$.

Interferences. In this paper, we do not restrict the study to perfect channels, and instead consider that they may subject to *interferences* from the environment. Interferences are modelled as potential evolution of channel contents without a change in the global state of the system. As in [37], we model interferences by a preorder over words $\succeq \subseteq \Sigma^* \times \Sigma^*$, which will parametrise the semantics of dialogue systems. We denote by $w \succeq w'$ if w and w' are the contents of the buffer respectively before and after the interferences.

▶ **Definition 4** (Interference model). *(from [37]) An interference model is a binary relation $\succeq \subseteq \Sigma^* \times \Sigma^*$ satisfying the following axioms:*

Reflexivity	*Transitivity*	*Additivity*	*Integrity*	*Non-expansion*
$\dfrac{a \in \Sigma}{a \succeq a}$	$\dfrac{w \succeq w' \; w' \succeq w''}{w \succeq w''}$	$\dfrac{w_1 \succeq w_1' \; w_2 \succeq w_2'}{w_1 \cdot w_2 \succeq w_1' \cdot w_2'}$	$\dfrac{\varepsilon \succeq w}{w = \varepsilon}$	$\dfrac{w \succeq w'}{\lvert w\rvert \geq \lvert w'\rvert}$

Axiom *Additivity* defines that failures can happen at any part of the words; axiom *Integrity* says ε is the least word; and axiom *Non-expansion* says that $\succeq$ preserves the size of words. Based on interferences, we define three failures as follows:

- *Lossiness*: Possible leaks of messages during transmission are modelled by adding the axiom $a \succeq \varepsilon$.

- *Corruption*: Possible transformation of a message a into a message b is modelled by adding the axiom $a \succeq b$.
- *Out-of-order*: Out-of-order communications are modelled by adding axioms $a \cdot b \succeq b \cdot a$ for all $a, b \in \Sigma$.

We now define successor configurations for FIFO systems with interferences.

▶ **Definition 5** (Successor configuration under interference). *Let $\mathcal{S}$ be a FIFO system. A configuration $\gamma' = (\vec{q}'; \vec{w}')$ is a* successor *of another configuration $\gamma = (\vec{q}; \vec{w})$, by firing the transition $(q_\mathsf{p}, a, q'_\mathsf{p}) \in \delta_\mathsf{p}$, written $\gamma \to \gamma'$ or $\gamma \xrightarrow{a} \gamma'$, if either:* (1) *$a = \mathsf{pq}!m$ and* (a) *$q'_\mathsf{r} = q_\mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$; and* (b) *$w'_\mathsf{pq} \preceq w_\mathsf{pq} \cdot m$ and $w'_\mathsf{rs} \preceq w_\mathsf{rs}$ for all $\mathsf{rs} \neq \mathsf{pq}$; or* (2) *$a = \mathsf{qp}?m$ and* (a) *$q'_\mathsf{r} = q_\mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$; and* (b) *$m \cdot w'_\mathsf{qp} \preceq w_\mathsf{qp}$ and $w'_\mathsf{rs} \preceq w_\mathsf{rs}$ for all $\mathsf{rs} \neq \mathsf{qp}$.*

The condition (1-b) puts the content to a channel pq, while (2-b) gets the content from a channel pq. The reflexive and transitive closure of $\to$ is $\xrightarrow{*}$. We write $\gamma_1 \xrightarrow{a_1 \cdot a_2 \cdots a_m} \gamma_{m+1}$ for $\gamma_1 \xrightarrow{a_1} \gamma_2 \cdots \gamma_m \xrightarrow{a_m} \gamma_{m+1}$. Moreover, we write $(\gamma_1, a_1 \cdot a_2 \cdots a_m, \gamma_{m+1}) \subseteq \delta$ to denote $\{(\gamma_1, a_1, \gamma_2), \ldots, (\gamma_m, a_m, \gamma_{m+1})\} \subseteq \delta$. A configuration γ is *reachable* if $\gamma_0 \xrightarrow{*} \gamma$ and we define $RS(\mathcal{S}) = \{\gamma \mid \gamma_0 \xrightarrow{*} \gamma\}$.

A configuration $\gamma = (\vec{q}; \vec{w})$ is said to be *k-bounded* if for all $\mathsf{pq} \in Ch$, $|w_\mathsf{pq}| \leq k$. We say that an execution $e = e_1 e_2 \ldots e_n$ is *k-bounded* from γ_1 if $\gamma_1 \xrightarrow{e_1} \gamma_2 \ldots \gamma_n \xrightarrow{e_n} \gamma_{n+1}$ and for all $1 \leq i \leq n+1$, γ_i is k-bounded; we denote this as $\gamma_1 \xrightarrow{e}_k \gamma_{n+1}$.

We define the k-reachability set of $\mathcal{S}$ to be the largest subset $RS_k(\mathcal{S})$ of $RS(\mathcal{S})$ within which each configuration γ can be reached by a k-bounded execution from γ_0. Note that, given a FIFO system $\mathcal{S}$, for every integer k, the set $RS_k(\mathcal{S})$ is finite and computable.

▶ **Example 6.** Let us revisit the system in Figure 2 and explore each of the interferences with the following executions (we denote by red the messages subject to interference):

- *Corruption*: Let us consider execution $e_c = \mathsf{sr}!start \,.\, \mathsf{sr}?start \,.\, \mathsf{sr}!data \,.\, \mathsf{sr}?end \,.\, \mathsf{rs}!ack \,.\mathsf{sr}!data$. Here, the message *data* has been corrupted to *end*. Hence, process r incorrectly receives the message *end*, and assumes that process s has stopped sending *data*, while process s continues to send it.
- *Lossiness*: Consider the execution $e_\ell = \mathsf{sr}!start \,.\, \mathsf{sr}?start \,.\, \mathsf{sr}!data \,.\, \mathsf{sr}?data \,.\, \mathsf{sr}!end$. Here, the message *end* has been lost, which means process r will be stuck waiting for process s to either send *data* or *end*.
- *Out-of-order*: Let $e_o = \mathsf{sr}!start \,.\, \mathsf{sr}?start \,.\, \mathsf{sr}!data \,.\, \mathsf{sr}!end \,.\, \mathsf{sr}?end \,.\, \mathsf{rs}!ack \,.\, \mathsf{rs}?ack \,.\, \mathsf{sr}?data \,.\, \mathsf{rs}!err \,.\, \mathsf{rs}?err$. In this case, the order of *data* and *end* has been swapped in the queue, which leads to a configuration where the error message is sent.

As shown in [1, 21], for communicating automata with lossiness, the reachability set is recognisable, and the reachability problem is decidable. In the case of out-of-order scheduling, it is easy to see that the problem reduces to reachability in Petri nets. It is less clear, but it can also be reduced to Petri net reachability problem in case of corruption. However, the complexity of reachability for these systems is very high – it is non-primitive recursive for lossy systems [44], and Ackermann-hard for corruption and out-of-order [13]. Hence, it is still worth exploring subclasses in the presence of errors.

3 RSC systems with interferences

We first extend the definitions of synchrony in systems from [18] to consider possible interferences. The main extension relates to the definition of *matching pairs*. Intuitively, matching pairs refer to a send action and the corresponding receive action in a given execution.

In the presence of interferences, it is not necessary that the same message that is sent is received (due to corruption), or that the k-th send action corresponds to the k-th receive action (due to lossiness or out-of-order). Hence we extend the definition of matching pairs.

▶ **Definition 7** (Matching pair with interference). *Given an execution $e = a_1 \dots a_n$, if there exists a channel* pq*, messages $m, m' \in \Sigma$ and $j, j', k, k' \in \{1, \dots, n\}$ where $j < j'$, and the following four conditions:*

(1) $a_j = \mathsf{pq}!m$; (2) $a_{j'} = \mathsf{pq}?m'$; (3) a_j *is the k-th send action to* pq *in e; and* (4) $a_{j'}$ *is the k'-th receive action on* pq *in e, then we say that $\{j, j'\} \subseteq \{1, \dots, n\}$ is a* matching pair with interference*, or i-matching pair.*

Note that if $m = m'$ and $k = k'$, we are back to the original definition of matching pairs in [17, Section 2], which we shall refer to henceforth as *perfect matching pairs*. When we refer to a matching pair, we mean either a perfect or i-matching pair. Moreover, our formalism allows for a single message to have more than one kind of interference, e.g. the same message can be corrupted and received out-of-order.

▶ **Example 8.** Consider the following execution $e = a_1 \dots a_5 = \mathsf{pq}!a \cdot \mathsf{qp}!b \cdot \mathsf{qp}?b \cdot \mathsf{pq}!c \cdot \mathsf{pq}?c$. For the channel qp, we have a perfect matching pair $\{2, 3\}$ which corresponds to the actions $\mathsf{qp}!b$ and $\mathsf{qp}?b$, the 1st send and receive action along qp. For the channel pq, we see that the 1st receive action is not $\mathsf{pq}?a$, and hence, there is no perfect matching pair corresponding to $\mathsf{pq}!a$. However, in case of interferences, we can have the following cases:

- If the message a is lost, i.e., $\mathsf{pq}!a$ would be a lost action, $\mathsf{pq}!c \cdot \mathsf{pq}?c$ would be a matched send-receive pair, and therefore, $\{4, 5\}$ would be the corresponding i-matching pair.
- If the message a was corrupted to c, then, $\mathsf{pq}?c$ would be the receive action corresponding to $\mathsf{pq}!a$, and we would have $\{1, 5\}$ as an i-matching pair.
- If the trace with an appended action as follows: $e' = \mathsf{pq}!a \cdot \mathsf{qp}!b \cdot \mathsf{qp}?b \cdot \mathsf{pq}!c \cdot \mathsf{pq}?c \cdot \mathsf{pq}?a$, then it could be that messages a and c were scheduled out-of-order in the channel pq. Then we have i-matching pairs $\{1, 6\}$ and $\{4, 5\}$.

We now modify the definition of *interactions* from [17] as follows.

▶ **Definition 9** (Interaction). *An* interaction *of e is either a (perfect or i-) matching pair, or a singleton $\{j\}$ such that a_j is a send action and j does not belong to any matching pair (such an interaction is called unmatched send).*

Given $e = a_1 \cdots a_n$, a set of interactions ν is called a *valid communication* of e if for every index $j \in \{1, \dots, n\}$, there exists exactly one interaction $\chi \in \nu$ such that $j \in \chi$. I.e., we need to ensure that every action in e belongs to exactly one interaction in the valid communication. We denote by $\mathsf{Comm}(e)$ the set of all valid communications associated to e.

▶ **Example 10.** Revisiting Example 8, given the execution $e = a_1 \dots a_5 = \mathsf{pq}!a \cdot \mathsf{qp}!b \cdot \mathsf{qp}?b \cdot \mathsf{pq}!c \cdot \mathsf{pq}?c$, there are two valid communications, $\nu_1 = \{\{1, 5\}, \{2, 3\}, \{4\}\}$ and $\nu_2 = \{\{1\}, \{2, 3\}, \{4, 5\}\}$, and $\mathsf{Comm}(e) = \{\nu_1, \nu_2\}$.

For the rest of this section, when we refer to an execution, we are referring to a tuple (e, ν) such that $\nu \in \mathsf{Comm}(e)$. We say that two actions a_1, a_2 *commute* if $\mathsf{proc}(a_1) \neq \mathsf{proc}(a_2)$ and they do not form a matching pair.

Given an execution (e, ν) such that $e = a_1 \dots a_n$ and $\nu \in \mathsf{Comm}(e)$, we say that $j \prec_{e,\nu} j'$ if (1) $j < j'$ and (2) a_j, $a_{j'}$ do not commute in ν. We now graphically characterise *causally equivalent* executions, using the notion of a conflict graph. This allows us to establish equivalences between different executions in which actions can be interchanged.

▶ **Definition 11** (Conflict graph). *Given an execution (e, ν), the conflict graph* $\mathsf{cgraph}(e, \nu)$ *is the directed graph $(\nu, \rightarrow_{e,\nu})$ where for all interactions $\chi_1, \chi_2 \in \nu$, $\chi_1 \rightarrow_{e,\nu} \chi_2$ if there is $j_1 \in \chi_1$ and $j_2 \in \chi_2$ such that $j_1 \prec_{e,\nu} j_2$.*

The conflict graph corresponding to Example 10, $\mathsf{cgraph}(e, \nu_1)$, is:



Two executions (e, ν) and (e', ν') are causally equivalent, denoted by $(e, \nu) \sim (e', \nu')$, if their conflict graphs are isomorphic.

We are now ready to define i-RSC systems, which is the extension of RSC to include interferences. Intuitively, i-RSC executions can be reordered to mimic rendezvous (or synchronous) communication. In the case with interference, we enforce that every valid communication is equivalent to a RSC execution.

▶ **Definition 12** (i-RSC system). *An execution (e, ν) is i-RSC if all matching pairs in ν are of the form $\{j, j+1\}$. A system $\mathcal{S}$ is i-RSC if for all tuples (e, ν) such that $e \in \mathsf{executions}(\mathcal{S})$ and $\nu \in \mathsf{Comm}(e)$, we have $\mathsf{cgraph}(e, \nu) = \mathsf{cgraph}(e', \nu')$ where (e', ν') is an i-RSC execution.*

▶ **Example 13.** From Ex. 10, (e, ν_2) is an i-RSC execution, but (e, ν_1) is neither an i-RSC execution nor equivalent to one, as message a has to be sent before message b is received by process p while message b has to be sent before the corresponding message (which is now c due to corruption) is received by process q.

This is the strictest version, however, this can be adapted to include only one communication by assuming a single instance of ν instead of all. We formalise our observation about non-i-RSC behaviours from Example 13, and show that i-RSC still maintains the good properties of the conflict graph as in [17].

▶ **Lemma 14.** *An execution (e, ν) is causally equivalent to an i-RSC execution iff the associated conflict graph* $\mathsf{cgraph}(e, \nu)$ *is acyclic.*

A borderline violation for interferences defined below is a key concept for the decidability of RSC systems. Intuitively, it provides a "minimal counter-example" for non-RSC behaviour.

▶ **Definition 15** (Borderline violation). *An execution (e, ν) is a borderline violation if* (1) *(e, ν) is not causally equivalent to an i-RSC execution,* (2) *$e = e' \cdot \mathsf{c}?m$ for some execution e' such that* (a) *for all $\nu' \in \mathsf{Comm}(e')$, (e', ν') is equivalent to an i-RSC execution and* (b) *there exists $\nu_1 \in \mathsf{Comm}(e')$ such that (e', ν_1) is an i-RSC execution.*

▶ **Lemma 16.** *$\mathcal{S}$ is i-RSC if and only if for all $e \in \mathsf{executions}(\mathcal{S})$ and $\nu \in \mathsf{Comm}(e)$, (e, ν) is not a borderline violation.*

Following the same approach as in [17], we show that inclusion into the i-RSC class is decidable. For simplicity, we construct the following sets: $\mathsf{Act}_{nr} = \{\mathsf{c}!?m \mid \mathsf{c}!m \in \mathsf{Act}, \mathsf{c}?m' \in \mathsf{Act}\} \cup \{\mathsf{c}!m \mid \mathsf{c}!m \in \mathsf{Act}\}$ and $\mathsf{Act}_? = \{\mathsf{c}?m \mid \mathsf{c}?m \in \mathsf{Act}\}$. Note that $\mathsf{c}!?m$ could include a send-receive pair where the message sent is different from the one received. This ensures inclusion of matching pairs due to corruption. An i-RSC execution can be represented by a word in Act_{nr}^* and a borderline violation by a word in $\mathsf{Act}_{nr}^*.\mathsf{Act}_?$. We first show that the set of borderline violations is regular.

▶ **Lemma 17.** *Let $\mathcal{S}$ with $\mathsf{product}(\mathcal{S}) = (Q, \Sigma, Ch, \mathsf{Act}, \delta, q_o)$. There is a non-deterministic finite state automaton $\mathcal{A}_{bv}$ computable in time $\mathcal{O}(|Ch|^3|\Sigma|^2)$ such that $\mathcal{L}(\mathcal{A}_{bv}) = \{e \in \mathsf{Act}_{nr}^*.\mathsf{Act}_? \mid \exists \nu \in \mathsf{Comm}(e)$ such that (e, ν) is a borderline violation$\}$.*

Next we show that the subset of executions $\mathsf{executions}(\mathcal{S})$ that begin with an i-RSC prefix and terminate with a reception is regular. The construction of the automaton $\mathcal{A}_{rsc}$ recognising such a language mimics the i-RSC executions of the original system $\mathcal{S}$, storing only the information on non-empty buffers, guessing which is the send message that will be matched by the final reception.

For the following result, we let the size $|\mathcal{A}|$ of an automaton $\mathcal{A} = (Q, \delta, q_0)$ be $|Q| + |\delta|$. Moreover, the size $|\mathcal{S}|$ of a system $\mathcal{S} = (\mathcal{A}_{\mathsf{p}})_{\mathsf{p}\in\mathbb{P}} = \sum_{\mathsf{p}\in\mathbb{P}} |\mathcal{A}_{\mathsf{p}}|$.

▶ **Lemma 18.** *Let $\mathcal{S}$ be a FIFO system. There exists a non-deterministic finite state automaton $\mathcal{A}_{rsc}$ over $\mathsf{Act}_{nr} \cup \mathsf{Act}_?$ such that $\mathcal{L}(\mathcal{A}_{rsc}) = \{e \cdot \mathsf{pq}?m \in \mathsf{Act}_{nr}^*.\mathsf{Act}_? \mid e \cdot \mathsf{pq}?m \in \mathsf{executions}(\mathcal{S})$ and $\exists \nu \in \mathsf{Comm}(e)$ such that (e, ν) is an i-RSC execution$\}$, which can be constructed in time $\mathcal{O}(n^{|\mathbb{P}|+2}|Ch|^2 \times 2^{|Ch|})$, where n is the size of $\mathcal{S}$.*

Using the above lemmas, we derive the following main theorem in this section, which states that the inclusion of an i-RSC system is decidable; and the complexity is comparable to that of checking inclusion to RSC [17, Theorem 12].

▶ **Theorem 19.** *Given a system $\mathcal{S}$ of size n, deciding whether it is an i-RSC system can be done in time $\mathcal{O}(n^{|\mathbb{P}|+2}|Ch|^5 \times 2^{|Ch|} \times |\Sigma|^2)$.*

4 k-Multiparty Compatibility with interferences

We extend our analyses to consider *k-multiparty compatibility* (k-MC) which was introduced in [35] for a subset of FIFO systems, called *communicating session automata* (CSA). CSA strictly include systems corresponding to asynchronous multiparty session types [15].

▶ **Definition 20** (Communicating session automata). *A deterministic FIFO automaton which has no mixed states is defined as a* session automaton. *FIFO systems comprising session automata are referred to as communicating session automata (CSA).*

In this section, we only consider communicating session automata. We begin by recalling the definition of k-MC which is composed of two properties, k-safety and k-exhaustivity.

▶ **Definition 21** (k-Safety, Definition 4 in [35]). *A communicating system $\mathcal{S}$ is* k-safe *if the following holds for all $(\vec{q}; \vec{w}) \in RS_k(\mathcal{S})$:*

(k-ER) $\forall \mathsf{pq} \in Ch$, *if* $\vec{w}_{\mathsf{pq}} = m.u$ *then* $(\vec{q}; \vec{w}) \xrightarrow{*}_k \xrightarrow{\mathsf{pq}?m}_k$.

(k-PG) *if q_{p} is receiving, then* $(\vec{q}; \vec{w}) \to_k \xrightarrow{\mathsf{pq}?m}$ *for some* $m \in \Sigma$.

The k-safety condition is composed of two properties, the first being eventual reception (k-ER) which ensures that every message sent to a channel is eventually received. The other property is progress (k-PG) where the system is not "stuck" at any receiving state.

A system is k-exhaustive if for all k-reachable configurations, whenever a send action is enabled, it can be fired within a k-bounded execution.

▶ **Definition 22** (k-Exhaustivity, Definition 8 in [35]). *A communicating system $\mathcal{S}$ is k-exhaustive if for all $(\vec{q}; \vec{w}) \in RS_k(\mathcal{S})$ and $\mathsf{pq} \in Ch$, if q_{p} is a sending state, then for all $(q_{\mathsf{p}}, \mathsf{pq}!m, q'_{\mathsf{p}}) \in \delta_{\mathsf{p}}$, there exists* $(\vec{q}; \vec{w}) \xrightarrow{*}_k \xrightarrow{\mathsf{pq}!m}_k$.

Example 23 shows that the reachability set of k-MC is not necessarily regular, unlike the binary half-duplex systems [14].

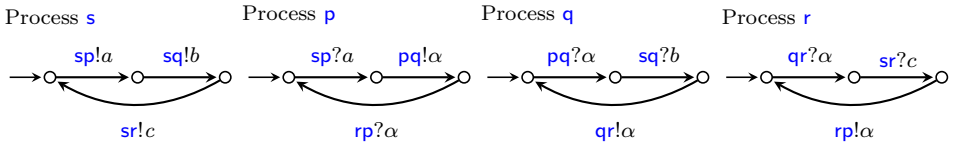


Figure 3 The above system $\mathcal{S}$ is k-MC but has a non-regular reachability set.

▶ **Example 23.** The system $\mathcal{S}$ depicted in Figure 3 is an example of a k-MC system for which the reachability set is not regular. It consists of four participants, sending messages amongst themselves. The first participant s can send equal number of a, b, c letters in a loop to participants p, q, and r respectively. The participants p, q, and r behave similarly, so let us take the example of p. It consumes one letter from the channel sp, then as a way of synchronisation sends a message α to q and waits to receive a message α from r. This ensures that participants p, q, and r consume equal number of letters from their respective channels with s. Hence, the reachability set for initial configuration (s_0, p_0, q_0, r_0) is $a^n\#b^n\#c^n$ which is context-sensitive, hence non-regular.

We prove that k-MC in the absence of errors, for a large class of systems the k-safety property subsumes k-exhaustivity.

▶ **Theorem 24.** *If a directed CSA $\mathcal{S}$ is k-safe, then $\mathcal{S}$ is k-exhaustive.*

***k*-MC with interferences.** Theorem 24 shows that the k-safety is a too strong condition in the presence of interferences. For instance, in case of lossiness, progress cannot be guaranteed. This is because there is always the potential of losing messages and being in a receiving state forever. We are now ready to define k-weak multiparty compatibility.

▶ **Definition 25** (k-Weak Multiparty Compatibility)**.** *A communicating system $\mathcal{S}$ is* weakly k-MC*, or k-WMC, if it satisfies k-ER and is k-exhaustive.*

This notion covers a larger class of systems than k-MC systems, and it is more natural in the presence of errors. Moreover, we still retain the decidability of k-WMC in the presence of errors. We briefly discuss weaker refinements to these properties in § 8. We conclude with the following theorem which states that the k-WMC property is decidable.

▶ **Theorem 26.** *Given a system $\mathcal{S}$ with lossiness (resp. corruption, resp. out-of-order) errors, checking the k-WMC property is decidable and* PSPACE*-complete.*

Proof. To check whether $\mathcal{S}$ is not k-exhaustive, i.e., for each sending state q_{p} and send action from q_{p}, we check whether there is a reachable configuration from which this send action cannot be fired. Hence, we need to search $RS_k(\mathcal{S})$, which has an exponential number of configurations (wrt. k). Note that due to interferences, each of these configurations can now have modified channel contents. We need to store at most $|\mathbb{P}|^n|Ch||\Sigma|^k$ configurations, where n is the maximum number of local states of a FIFO automata, following ideas from [35] and [9]. Hence, the problem can be decided in polynomial space when k is given in unary.

Next, to show that k-WER is decidable, we check for every such reachable configuration, that there exists a receive action from the same channel (note that we do not need to ensure it is the same message). ◀

5 Crash-stop failures

Session types [27, 28, 46] are a type discipline to ensure communication safety for message passing systems. Most session types assume a scenario where participants operate reliably, i.e. communication happens without failures. To model systems closer to the real world, Barwell et al. [3, 6] introduced session types with *crash-stop failures*. In this section, we consider the same notion for communicating systems which we define as crash-handling.

5.1 Crash-handling FIFO systems

We extend this framework to FIFO systems. As in [3, 6], we declare a (potentially empty) set of *reliable processes*, which we denote as $\mathcal{R} \subseteq \mathbb{P}$. If a process is assumed reliable, the other processes can interact with it without needing to handle its crashes. Hence if $\mathcal{R} = \mathbb{P}$, there is no additional crash-handling behaviour for the system. In this way, we can model a mixture of reliable and unreliable processes. For simplicity in the construction, we enforce an additional constraint that in the crash-handling branches, there is no receive action from the crashed process.

We use a shorthand for the *broadcast* of a message $m \in \Sigma$ by process $\mathsf{p} \in \mathbb{P}$ along all outgoing channels: $(q, \mathsf{broadcast}_{\mathsf{p}}(m), q')$ if $q \xrightarrow{\mathsf{pr}_1!m.\mathsf{pr}_2!m \ldots \mathsf{pr}_n!m} q'$ such that $n = |Ch_{\mathsf{o},\mathsf{p}}|$ and $\mathsf{r}_\mathsf{i} \neq \mathsf{r}_\mathsf{j}$ for all $\mathsf{i} \neq \mathsf{j}$. We denote by $\mathsf{crash\text{-}broadcast}_{\mathsf{p}}(m)$ the concatenation $\mathit{crash}_{\mathsf{p}} \cdot \mathsf{broadcast}_{\mathsf{p}}(m)$ where $\mathit{crash}_{\mathsf{p}} \in \mathsf{Act}_\tau$ is an internal action reserved for when process p crashes.

Let $\mathcal{S} = (\mathcal{A}_\mathsf{p})_{\mathsf{p} \in \mathbb{P}}$ be a FIFO system over $\Sigma \uplus \{\lightning\}$. Let the set of reliable processes be $\mathcal{R} \subseteq \mathbb{P}$. For each $\mathsf{p} \in \mathbb{P}$:

- We divide the state set as follows : $Q_\mathsf{p} = Q_{\mathsf{p},1} \uplus Q_{\mathsf{p},2} \uplus Q_{\mathsf{p},3}$.
- Let $\mathsf{Act} \subseteq (Ch \times \{!, ?\} \times (\Sigma \uplus \{\lightning\})) \uplus \mathsf{Act}_\tau$ be the set of actions.
- We split $\delta_\mathsf{p} = \delta_{\mathsf{p},1} \uplus \delta_{\mathsf{p},2}$ such that:
  - $\delta_{\mathsf{p},1} \subseteq Q_{\mathsf{p},1} \times (Ch \times \{!, ?\} \times \Sigma) \times (Q_{\mathsf{p},1} \cup Q_{\mathsf{p},2})$, and
  - $\delta_{\mathsf{p},2} \subseteq Q_\mathsf{p} \times [(Ch \times \{!, ?\} \times \{\lightning\}) \cup \mathsf{Act}_\tau] \times Q_\mathsf{p}$.

We say that a process p has crash-handling behaviour in $\mathcal{S}$ if $\delta_{\mathsf{p},2}$ is the smallest set of transitions such that:

1. *Crash handling* (CH): For all $(q, \mathsf{rp}?a, q') \in \delta_{\mathsf{p},1}$ such that $\gamma_0 \xrightarrow{e} \gamma = (\vec{q}; \vec{w})$ and $\vec{q}_\mathsf{p} = q$ and $\mathsf{r} \in \mathbb{P} \setminus \mathcal{R}$, there exists $q'' \in (Q_{\mathsf{p},1} \cup Q_{\mathsf{p},2})$ such that $(q, \mathsf{rp}?\lightning, q'') \in \delta_{\mathsf{p},2}$.
2. *Crash broadcast* (CB): If $\mathsf{p} \notin \mathcal{R}$, then for all $q \in Q_{\mathsf{p},1}$, there exists a crash-broadcast $(q, \mathsf{crash\text{-}broadcast}_{\mathsf{p}}(\lightning), q_{\mathsf{stop}}) \subseteq \delta_{\mathsf{p},2}$, for some $q_{\mathsf{stop}} \in Q_{\mathsf{p},2}$ and all intermediate states belonging to $Q_{\mathsf{p},3}$.
3. *Crash redundancy* (CR): Finally, we have the condition that any dangling crash messages are cleaned up. For all $q \in Q_{\mathsf{p},2}$, $(q, \mathsf{rp}?\lightning, q) \in \delta_{\mathsf{p},2}$.

Condition (CH) enforces that every state in the system which receives from an unreliable process has a crash-handling branch, so that the receiving process is not deadlocked waiting for a message from a process that has crashed. Condition (CB) ensures that every unreliable process can non-deterministically take the internal action $\mathit{crash}_\mathsf{p}$ when it crashes and broadcast this information to all the other participants. Condition (CR) ensures that from all states in $Q_{\mathsf{p},2}$, any dangling crash messages are cleaned up from an (otherwise empty) channel.

▶ **Definition 27** (Crash-handling systems). *We say that a system $\mathcal{S}$ is crash-handling if every process $\mathsf{p} \in \mathbb{P}$ has crash-handling behaviour in $\mathcal{S}$.*

Consider the following example, which models a simple send-receive protocol between a sender and a receiver.

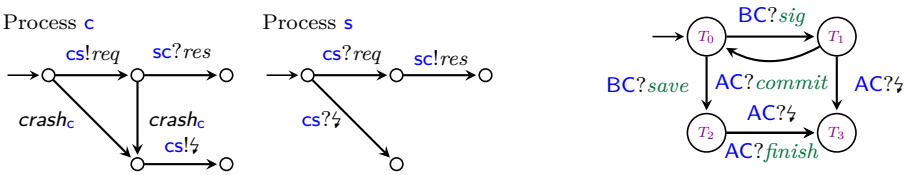


Figure 4 (a) The system $\mathcal{S}$ (right) is crash-handling. (b) FIFO automata (right) of the type in Example 36.

▶ **Example 28.** Figure 4(a) shows a crash-handling system. It consists of two processes, a server (s) and a client (c). We assume that the server is reliable, i.e. does not crash, while the client is unreliable, i.e could crash. Hence the client can crash in any control state, while the server is always ready to handle a crash when it is waiting for a message from the client.

In this construct, it is still possible to send messages to a crashed process. This is because from the perspective of the sending process, the crash of the receiving process is not necessarily known. Therefore, in this model, while processes can continue to send messages to crashed processes, the crashed processes would not be able to receive any messages.

Note that these properties are local to each individual automaton, hence the verification of these properties is decidable.

▶ **Lemma 29.** *It is decidable to check whether a system is crash-handling.*

We see that this behaviour can be appended to any FIFO system, but it does not affect the underlying verification properties of the automata. We demonstrate with the example of boundedness (i.e. checking if every execution is k-bounded for some k), but a similar argument can be used for reachability or deadlock.

▶ **Lemma 30.** *The boundedness problem is undecidable for crash-handling systems.*

5.2 Crash-handling subsystems

Next we investigate inclusion of crash-handling systems in the aforementioned classes.

Crash-handling RSC systems. Checking that the RSC property is decidable for crash-handling systems amounts to verifying if the proofs hold for communicating automata with internal actions. Let us first look at an example.

▶ **Example 31.** The system in Figure 4 is a crash-handling system that is also RSC. We see that the behaviour of the system in the absence of crashes is RSC, and in the presence of crashes, there is no additional non-RSC behaviour. Moreover, even if the *req* is sent, followed by the crash broadcast – since the crash message is never received, the behaviour of the system is still RSC. However, this need not be the case for other examples.

Next we show that the proofs from [17] can be adapted to automata with internal actions.

▶ **Theorem 32.** *Given a crash-handling system $\mathcal{S}$, it is decidable to check inclusion to the RSC class.*

Crash-handling k-WMC systems. We now show that checking k-WMC is decidable for crash-handling systems generated from a collection of local types. The reason for considering k-WMC instead of k-MC in [35, Definition 9] is that for crash-handling systems generated from local types, the end states are receiving states (as opposed to *final states*). This result is adapted from [35] with the inclusion of internal actions.

▶ **Theorem 33.** *Given a crash-handling system $\mathcal{S}$ generated from a collection of communicating session automata, it is decidable to check k-WMC, and can be done in* PSPACE.

6 Session types with crash-stop failures

This section shows that the crash-handling system strictly subsumes the crash-stop systems in [3, 5], preserving the semantics. We recall the crash-stop semantics for local types defined in [3] where the major additions are (1) a special local type stop to denote crashed processes; and (2) a crash-handling branch (*catch*) in one of branches of an external choice.

The syntax of the local types ($S, T, \ldots$) are given as:

$$\begin{array}{rcll} S,T & ::= & \mathsf{p}?\{m_i.T_i\}_{i\in I} \mid \mathsf{p}!\{m_i.T_i\}_{i\in I} & \text{(external choice, internal choice)} \\ & \mid & \mu\mathbf{t}.T \mid \mathbf{t} \mid \mathsf{end} \mid \mathsf{stop} & \text{(recursion, type variable, end, crash)} \end{array}$$

An external choice (branching) (resp., an internal choice (selection)), denoted by $\mathsf{p}?\{m_i.T_i\}_{i\in I}$ (resp., $\mathsf{p}!\{m_i.T_i\}_{i\in I}$) indicates that the *current* role is to *receive* from (or *send* to) the process p. We require pairwise-distinct, non-empty labels and the crash-handling label (*catch*) not appear in *internal* choices; and that singleton crash-handling labels not permitted in external choices. The type end indicates a *successful* termination (omitted where unambiguous), and recursive types are assumed *guarded*, i.e., $\mu\mathbf{t}.\mathbf{t}$ is not allowed, and recursive variables are unique. A *runtime* type stop denotes crashes.

We point out here that while this is a bottom-up view of the crash-handling behaviour introduced in [3], we have taken a purely type-based approach here. For a calculus based approach, we refer the reader to [4].

We define the LTS over local types and extend the notions to communicating systems. We use the same labels as the ones for communicating systems.

▶ **Definition 34** (LTS over local types). *The relation $T \xrightarrow{a} T'$ for the local type of role p is defined as:*

$$\begin{array}{lll} \text{[LR1]} & \mathsf{q}\dagger\{m_i.T_i\}_{i\in I} \xrightarrow{\mathsf{pq}\dagger m_k} T_k, & \text{where } \dagger \in \{!,?\} \text{ and } m_k \neq \mathit{catch}. \\ \text{[LR2]} & T[\mu\mathbf{t}.T/\mathbf{t}] \xrightarrow{a} T' \Longrightarrow \mu\mathbf{t}.T \xrightarrow{a} T' & \\ \text{[LR3]} & \mathsf{q}\dagger\{m_i.T_i\}_{i\in I} \xrightarrow{\mathit{crash\text{-}broadcast}_{\mathsf{p}}(\lightning)} \mathsf{stop}, & \text{where } \dagger \in \{!,?\}. \\ \text{[LR4]} & \mathsf{q}?\{m_i.T_i\}_{i\in I} \xrightarrow{\mathsf{qp}?\lightning} T_k, & \text{if } m_k = \mathit{catch}. \\ \text{[LR5]} & T \xrightarrow{\mathsf{qp}?\lightning} T, & \forall \mathsf{q} \in \mathbb{P} \setminus \{\mathsf{p}\} \text{ for } T \in \{\mathsf{stop}, \mathsf{end}\}. \end{array}$$

Rules [LR1] and [LR2] are standard output/input and recursion rules, respectively; rule [LR3] accommodates for the crash of a process; rule [LR4] is the main rule for crash-handling where the reception of crash information leads the process to a crash-handling branch; and rule [LR5] allows any dangling crash information messages to be read in the sink states.

The LTS over a set of local types is defined as in Definition 2, where a configuration $\gamma = (\vec{T}; \vec{w})$ of a system is a pair with $\vec{T} = \{T_{\mathsf{p}}\}_{\mathsf{p}\in\mathbb{P}}$ and $\vec{w} = (w_{\mathsf{pq}})_{\mathsf{pq}\in Ch}$ with $w_{\mathsf{pq}} \in \Sigma^*$.

Next we algorithmically translate from local types to FIFO automata preserving the trace semantics. Below we write $\mu\vec{\mathbf{t}}.T$ for $\mu\mathbf{t}_1.\mu\mathbf{t}_2\ldots\mu\mathbf{t}_n.T$ with $n \geq 0$.

In order to construct the FIFO automata, we first need to define the set of states. Intuitively, this is the set of types which result from any continuation of the initial local type. Below we define *a type occurring in another type* (based on the definition in [47]).

▶ **Definition 35** (Type occurring in type, [47]). *We say a type T' occurs in T (denoted by $T' \in T$) if and only if at least one of the following conditions holds:* (1) *if T is $\mathsf{p}?\{m_i.T_i\}_{i\in I}$, there exists $i \in I$ such that $T' \in T_i$;* (2) *if T is $\mathsf{p}!\{m_i.T_i\}_{i\in I}$, there exists $i \in I$ such that $T' \in T_i$;* (3) *if T is $\mu\mathbf{t}.T_\mu$ then $T' \in T_\mu$; or* (4) *$T' = T$, where $=$ denotes the syntactic equality.*

▶ **Example 36.** Let $\mathbb{P} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$ and $\mathcal{R} = \{\mathsf{B}, \mathsf{C}\}$. Consider a local type of C: $T = \mu\mathbf{t}.\mathsf{B}?\{\mathit{sig}.\mathsf{A}?\{\mathit{commit}.\mathbf{t}, \mathit{catch}.\mathsf{end}\},\ \mathit{save}.\mathsf{A}?\{\mathit{finish}.\mathsf{end}, \mathit{catch}.\mathsf{end}\}\}$.

Then, the set of all $T' \in T$ is $\{T, \mathsf{B}?\{\mathit{sig}.\mathsf{A}?\{\mathit{commit}.\mathbf{t}, \mathit{catch}.\mathsf{end}\}\}, \mathsf{A}?\{\mathit{commit}.\mathbf{t}\}, \mathsf{B}?\{\mathit{save}.\mathsf{A}?\{\mathit{finish}.\mathsf{end}, \mathit{catch}.\mathsf{end}\}\}, \mathsf{A}?\{\mathit{catch}.\mathsf{end}\}, \mathsf{A}?\{\mathit{finish}.\mathsf{end}\}, \mathsf{end}, \mathbf{t}\}$.

And now, we are ready to define the FIFO automata.

▶ **Definition 37** (From local types to FIFO automata). *Let T_0 be the local type of participant p. The automaton corresponding to T_0 is $\mathcal{A}(T_0) = (Q, \delta, q_0)$ where:*

1. $Q = \{T' \mid T' \in T_0, T' \neq \mathbf{t}, T' \neq \mu\mathbf{t}.T\} \cup \{q_{\mathit{crash}}\} \cup \{q_{\mathit{send},\mathsf{r}} \mid \mathsf{r} \in \mathbb{P} \setminus \{\mathsf{p}\}\}$
2. $q_0 = \mathrm{strip}(T_0)$;
3. *δ is the smallest set of transitions such that $\forall T \in Q$:*
   - **a.** *If $T = \mathsf{q}\dagger\{m_i.T_i\}_{i\in I}$ and $k \in I$, $m_k \neq \mathit{catch}$, and $\dagger \in \{!, ?\}$*
   $$\begin{cases} (T, \mathsf{pq}\dagger m_k, \mathrm{strip}(T_k)) \in \delta & \text{if } T_k \neq \mathbf{t} \\ (T, \mathsf{pq}\dagger m_k, \mathrm{strip}(T')) \in \delta & \text{if } T_k = \mathbf{t} \text{ with } \mu\mathbf{t}.T' \in T_0. \end{cases}$$
   - **b.** *If $T = \mathsf{q}?\{m_i.T_i\}_{i\in I}$ with $k \in I$, $m_k = \mathit{catch}$*
   $$\begin{cases} (T, \mathsf{qp}?\lightning, \mathrm{strip}(T_k)) \in \delta & \text{if } T_k \neq \mathbf{t} \\ (T, \mathsf{qp}?\lightning, \mathrm{strip}(T')) \in \delta & \text{if } T_k = \mathbf{t} \text{ with } \mu\mathbf{t}.T' \in T_0. \end{cases}$$
   - **c.** *If $T \notin \{\mathsf{stop}, \mathsf{end}\}$, then $(T, \mathit{crash\text{-}broadcast}_{\mathsf{p}}(\lightning), \mathsf{stop}) \subseteq \delta$ where*
     - **i.** $(T, \mathit{crash}, q_{\mathit{crash}}) \in \delta$
     - **ii.** $(q_{\mathit{crash}}, \mathsf{pr}_1!\lightning, q_{\mathit{send},\mathsf{r}_1}) \in \delta$
     - **iii.** $(q_{\mathit{send},\mathsf{r}_i}, \mathsf{pr}_{i+1}!\lightning, q_{\mathit{send},\mathsf{r}_{i+1}}) \in \delta\ \forall i \in \{1, \ldots, n-2\}$, *where $n = |Ch_{o,\mathsf{p}}|$*
     - **iv.** $(q_{\mathit{send},\mathsf{r}_{n-1}}, \mathit{crash}, \mathsf{stop}) \in \delta$
   - **d.** *If $T \in \{\mathsf{stop}, \mathsf{end}\}$, then $(T, \mathsf{qp}?\lightning, T) \in \delta$ for all $\mathsf{q} \in \mathbb{P} \setminus \{\mathsf{p}\}$.*

where $\mathrm{strip}(T) \stackrel{\text{def}}{=} \mathrm{strip}(T')$ *if $T = \mu\mathbf{t}.T'$; otherwise* $\mathrm{strip}(T) \stackrel{\text{def}}{=} T$.

▶ **Example 38.** The FIFO automata constructed from the type in Example 36 is shown in Figure 4. We see that for all receiving actions from process A, which is not in the reliable set of processes, there is a crash-handling branch, where:

$$T_0 = \mathsf{B}?\{\mathit{sig}.\mathsf{A}?\{\mathit{commit}.\mathbf{t}, \mathit{catch}.\mathsf{end}\},\ \mathit{save}.\mathsf{A}?\{\mathit{finish}.\mathsf{end}, \mathit{catch}.\mathsf{end}\}\}$$
$$T_1 = \mathsf{A}?\{\mathit{commit}.\mathbf{t}, \mathit{catch}.\mathsf{end}\} \quad T_2 = \mathsf{A}?\{\mathit{finish}.\mathsf{end}, \mathit{catch}.\mathsf{end}\} \quad T_3 = \mathsf{end}$$

We now prove that the automata generated from a local type can be composed into communicating session automata, and this translation preserves the semantics.

▶ **Lemma 39.** *Assume T_{p} is a local type. Then $\mathcal{A}(T_{\mathsf{p}})$ is deterministic, directed and has no mixed states. Moreover, $T_{\mathsf{p}} \approx \mathcal{A}(T_{\mathsf{p}})$, i.e. $\forall \phi, \phi \in \mathsf{executions}(T_{\mathsf{p}}) \Leftrightarrow \phi \in \mathsf{executions}(\mathcal{A}(T_{\mathsf{p}}))$.*

By Lemma 39, we derive that the resulting systems belong to the class of crash-handling systems, and the problem of checking RSC and k-WMC is decidable for this class.

▶ **Theorem 40.** *The FIFO system generated from the translation of crash-stop session types is a crash-handling system. Moreover, it is decidable to check inclusion to the RSC and k-WMC classes.*

7 Experimental evaluation

Table 1 Experimental evaluation of benchmarks in the *ReSCu* and `kmc` tool, under the presence of no errors, out-of-order, lossiness and corruption errors. Note that the systems were checked for $k \leq 10$. TO denotes timeout after 5 minutes. The *-marked examples originate from the ReSCu tool [16], having been translated from other papers, as detailed in the original publication.

Protocol	No errors		Out of order		Lossiness				Corruption			
	k-MC	RSC	k-MC	RSC	k-exh	k-ER	k-PG	RSC	k-exh	k-ER	k-PG	RSC
Alternating Bit [41]	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	no	yes
Alternating Bit [7]	yes	no	yes	yes	yes	yes	no	yes	yes	yes	no	yes
Bargain [34]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Client-Server-Logger [35]	yes	no	yes	no	yes	yes	no	yes	yes	yes	no	yes
Cloud System v4 [25]	yes	yes	yes	no	no	no	no	yes	no	no	no	no
Commit protocol [10]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Dev System [40]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Elevator [10]	yes	no	yes	no	yes	yes	no	no	no	TO	no	no
Elevator-dashed [10]	yes	no	yes	no	no	no	no	no	no	TO	no	no
Elevator-directed [10]	yes	no	yes	no	no	no	no	no	no	TO	no	no
Filter Collaboration [49]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Four Player Game [34]	yes	yes	yes	no	no	yes	no	yes	no	yes	no	yes
Health System [35]	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes
Logistic [39]	yes	yes	yes	yes	yes	yes	no	yes	no	no	no	yes
Sanitary Agency (mod) [42]	yes	yes	yes	yes	yes	no	no	yes	yes	TO	no	yes
TPM Contract [26]	yes	yes	yes	no	yes	yes	no	yes	no	no	no	no
2-Paxos 2P3A ([45])	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes
Promela I* [16]	yes	no	yes	no	yes	yes	no	yes	yes	yes	yes	yes
Web Services* [16]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Trade System* [16]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Online Stock Broker* [16]	no	no	no	no	no	no	no	yes	no	no	no	yes
FTP* [16]	yes	yes	yes	yes	yes	yes	no	yes	no	no	no	yes
Client-server* [16]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Mars Explosion* [16]	yes	yes	yes	yes	yes	no	no	yes	no	no	no	yes
Online Computer Sale* [16]	no	yes	no	yes	yes	yes	no	yes	no	no	no	yes
e-Museum* [16]	yes	yes	yes	no	yes	no	no	yes	yes	no	no	yes
Vending Machine* [16]	yes	yes	yes	yes	yes	yes	no	yes	yes	no	no	yes
Bug Report* [16]	yes	yes	yes	no	yes	yes	no	yes	no	no	no	yes
Sanitary Agency* [16]	no	yes	no	yes	yes	yes	no	yes	yes	no	no	yes
SSH* [16]	no	yes	no	yes	yes	yes	no	yes	yes	yes	no	yes
Booking System* [16]	no	yes	no	yes	yes	yes	no	yes	yes	no	no	yes
Hand-crafted Example* [16]	no	yes	no	yes	yes	no	no	yes	yes	no	no	yes

We verify protocols in the literature under interferences in order to compare how inclusion to the RSC and k-WMC classes change; and which of RSC and k-MC is more resilient under failures. We used the tools, RSC-checker *ReSCu* [16] (implemented in OCaml), and k-MC-checker `kmc` [35] (implemented in Haskell). The *ReSCu* tool implements a version of the out-of-order scheduling with an option, so we use this available option to take our benchmark.

The `kmc` tool implements the options to check (1) the k-exhaustivity (k-EXH, Def. 22); (2) the k-eventual reception (k-ER, Def 21); and (3) the progress (k-PG, Def 21). The k-weak multiparty compatibility condition (k-WMC, Def 25) no longer checks for k-PG but checks k-EXH and k-ER. Hence checking (1,2,3) gives us the justification whether k-WMC is more resilient than k-MC. For the out-of-order, we implemented the out-of-order scheduling in Haskell mirroring the implementation as in *ReSCu*. To model lossiness, we add reception self-loops as defined in completely specified protocols [21], and for corruption, we allow the sending of arbitrary messages; both of these are implemented in Python.

Table 1 shows the evaluation results. From the benchmarks in [16], we selected all the relevant benchmarks (CSA and peer-to-peer) in order to evaluate them by `kmc`. Interestingly, in the case of out-of-order errors, all of the protocols which satisfy k-MC without errors still satisfy k-MC. However RSC does not satisfy some k-MC protocols. This would imply that in most real-world examples, the flexibility in behaviour introduced by relaxing the FIFO condition does not affect the inclusion to k-MC. On the other hand, under lossiness and corruption, most examples no longer belong to k-MC. More specifically, k-PG fails for most cases. The k-ER also fails for many cases, especially in the presence of corruption. This justifies a relevance of our definition of k-WMC under those two failures.

We implement a k-bounded version of the Paxos protocol [33], a consensus algorithm that ensures agreement in a distributed system despite failures like lossiness and reordering, using a process of proposing and accepting values (c.f. [45] for the details). This version limits retry attempts to k. Our implementation (for 2 retries, 2 proposers and 3 acceptors) shows it is k-MC and RSC both without errors, and k-MC under lossiness. Since Paxos does not assume corruption, it is unsurprising that it is no longer k-MC under corruption.

8 Conclusion and further related work

In this paper, we derived decidability and complexity results for two subclasses, RSC and k-MC, under two types of communication failures: interferences and crash-stop failures. In the absence of errors, RSC systems and k-MC systems are incomparable, even if we restrict the analyses to 1-MC systems. For example, [35, Example 4] is 1-MC but not RSC. Conversely, [17, Example 4] is RSC but does not satisfy the progress condition, and hence is not k-MC for any $k \in \mathbb{N}$. Despite these distinctions, both classes aim to generalise the concept of half-duplex communication to multiparty systems. This serves as our primary motivation for examining failures in a uniform way across both RSC and k-MC systems.

In the interference model, we introduced i-RSC systems, which relax the matching-pair conditions in RSC; and k-WMC which omits the progress condition to accommodate a model with no final states. We proved that the inclusion problem for these relaxed properties remain decidable within the same complexity class as their error-free counterparts. The evaluation results in § 7 confirm that relaxed systems are more resilient than the original ones.

As the second failure model, we investigated crash-stop failures. We defined crash-handling communicating systems which strictly include the class of local types with crash-stop failures. We also proved that both RSC and k-MC properties are decidable for this class. Note that multiparty session types with crash-stop failures studied in [6] are limited to *synchronous* communications. Meanwhile, the asynchronous setting in [3] restricts expressiveness to a set of local types projected from global types (which is known to be less expressive than those not using global types [43]). Therefore, both of these systems are strictly subsumed by our crash-handling system as proven in Theorem 40.

Integrating the k-MC-checker and the *ReSCu* tool (with support for crash-stop failures) into the Scala toolchain of [3] is a promising direction for future work, potentially enabling the verification of a broader class of programs than those considered in [3, 43].

Various failure-handling systems have been studied in the session types literature, e.g., affine session types [22, 29, 38], link-failure [2] and event-driven failures [48]. Interpreting their failures into our framework would offer a uniform analysis of failure processes.

References

1. P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 160–170, June 1993. doi:10.1109/LICS.1993.287591.
2. Manuel Adameit, Kirstin Peters, and Uwe Nestmann. Session Types for Link Failures. In Ahmed Bouajjani and Alexandra Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, pages 1–16, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-60225-7_1.
3. Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Designing Asynchronous Multiparty Protocols With Crash Stop Failures. In *European Conference on Object Oriented Programming*, volume 263, pages 1:1–1:30, 2023. doi:10.4230/LIPIcs.ECOOP.2023.1.
4. Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Designing Asynchronous Multiparty Protocols with Crash-Stop Failures, May 2023. arXiv:2305.06238 [cs]. doi:10.48550/arXiv.2305.06238.
5. Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Crash-Stop Failures in Asynchronous Multiparty Session Types. *Logical Methods in Computer Science*, 2025. URL: https://arxiv.org/abs/2311.11851.
6. Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou. Generalised Multiparty Session Types with Crash-Stop Failures. In Bartek Klin, S\lawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:25, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.CONCUR.2022.35.
7. Bernard Boigelot and Patrice Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 1–12, Berlin, Heidelberg, 1996. Springer. doi:10.1007/3-540-61474-5_53.
8. Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Etienne Lozes, and Amrita Suresh. A Unifying Framework for Deciding Synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.CONCUR.2021.14.
9. Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional Dynamic Logic for Message-Passing Systems. *Logical Methods in Computer Science*, Volume 6, Issue 3, September 2010. doi:10.2168/LMCS-6(3:16)2010.
10. Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the Completeness of Verifying Message Passing Programs Under Bounded Asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, Lecture Notes in Computer Science, pages 372–391, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-96142-2_23.
11. Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, April 1983. doi:10.1145/322374.322380.

12 Zak Cutner, Nobuko Yoshida, and Martin Vassor. Deadlock-free asynchronous message reordering in Rust with multiparty session types. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '22, pages 246–261, New York, NY, USA, March 2022. Association for Computing Machinery. doi:10.1145/3503221.3508404.

13 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The Reachability Problem for Petri Nets Is Not Elementary. *J. ACM*, 68(1):7:1–7:28, 2021. doi:10.1145/3422822.

14 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, November 2005. doi:10.1016/j.ic.2005.05.006.

15 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 174–186, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-39212-2_18.

16 Loïc Desgeorges and Loïc Germerie Guizouarn. RSC to the ReSCu: Automated Verification of Systems of Communicating Automata. In Sung-Shik Jongmans and Antónia Lopes, editors, *Coordination Models and Languages*, Lecture Notes in Computer Science, pages 135–143, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-35361-1_7.

17 Cinzia Di Giusto, Loïc Germerie Guizouarn, and Etienne Lozes. Multiparty half-duplex systems and synchronous communications. *Journal of Logical and Algebraic Methods in Programming*, 131:100843, February 2023. doi:10.1016/j.jlamp.2022.100843.

18 Cinzia Di Giusto, Loïc Germerie Guizouarn, and Etienne Lozes. Towards Generalised Half-Duplex Systems. *Electron. Proc. Theor. Comput. Sci.*, 347:22–37, October 2021. arXiv:2110.00145 [cs]. doi:10.4204/EPTCS.347.2.

19 Cinzia Di Giusto, Laetitia Laversa, and Etienne Lozes. On the k-synchronizability of Systems. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings*, Lecture Notes in Computer Science, pages 157–176, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45231-5_9.

20 Alan Fekete, Nancy Lynch, Yishay Mansour, and John Spinelli. The impossibility of implementing reliable communication in the face of crashes. *J. ACM*, 40(5):1087–1107, November 1993. doi:10.1145/174147.169676.

21 Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distrib Comput*, 7(3):129–135, March 1994. doi:10.1007/BF02277857.

22 Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional Asynchronous Session Types: Session Types without Tiers. *Proc. ACM Program. Lang.*, 3(POPL):28:1–28:29, 2019. doi:10.1145/3290341.

23 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On Communicating Automata with Bounded Channels. *Fundamenta Informaticae*, 80(1-3):147–167, January 2007. Publisher: IOS Press. URL: https://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09.

24 Blaise Genest, Anca Muscholl, and Dietrich Kuske. A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2004. doi:10.1007/978-3-540-30550-7_4.

25 Matthias Güdemann, Gwen Salaün, and Meriem Ouederni. Counterexample guided synthesis of monitors for realizability enforcement. In *ATVA 2012*, pages 238–253, 2012. doi:10.1007/978-3-642-33386-6_20.

26 Sylvain Hallé and Tevfik Bultan. Realizability analysis for message-based interactions using shared-state projections. In *SIGSOFT 2010*, pages 27–36, 2010. doi:10.1145/1882291.1882298.

27 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proceedings of ESOP 1998*, volume 1381 of *Lecture Notes in Computer Science*, pages 22–138. Springer, 1998.

28 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '08, pages 273–284, New York, NY, USA, January 2008. Association for Computing Machinery. doi:10.1145/1328438.1328472.

29 Ping Hou, Nicolas Lagaillardie, and Nobuko Yoshida. Fearless Asynchronous Communications with Timed Multiparty Session Protocols. In *ECOOP 2024*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ECOOP.2024.19.

30 Keigo Imai, Julien Lange, and Rumyana Neykova. Kmclib: Automated inference and verification of session types from OCaml programs. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 379–386. Springer, 2022. doi:10.1007/978-3-030-99524-9_20.

31 Shokoufeh Kazemlou and Borzoo Bonakdarpour. Crash-Resilient Decentralized Synchronous Runtime Verification. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 207–212, October 2018. ISSN: 2575-8462. doi:10.1109/SRDS.2018.00032.

32 Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Stay Safe under Panic: Affine Rust Programming with Multiparty Session Types. In *36th European Conference on Object-Oriented Programming*, volume 222 of *LIPIcs*, pages 4:1–4:29. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ECOOP.2022.4.

33 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998. doi:10.1145/279227.279229.

34 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*, pages 221–232, 2015. doi:10.1145/2676726.2676964.

35 Julien Lange and Nobuko Yoshida. Verifying Asynchronous Interactions via Communicating Session Automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 97–117, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-25540-4_6.

36 Markus Lohrey and Anca Muscholl. Bounded MSC Communication. 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 295–309. Springer, 2002. doi:10.1007/3-540-45931-6_21.

37 Etienne Lozes and Jules Villard. Reliable Contracts for Unreliable Half-Duplex Communications. In Marco Carbone and Jean-Marc Petit, editors, *Web Services and Formal Methods*, Lecture Notes in Computer Science, pages 2–16, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-29834-9_2.

38 Dimitris Mostrous and Vasco Thudichum Vasconcelos. Affine sessions. In *International Conference on Coordination Languages and Models*, pages 115–130, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-43376-8_8.

39 OMG. Business Process Model and Notation, 2018. https://www.omg.org/spec/BPMN/2.0/.

40 Roly Perera, Julien Lange, and Simon J. Gay. Multiparty compatibility for concurrent objects. In *PLACES 2016*, pages 73–82, 2016. doi:10.4204/EPTCS.211.8.

41 Introduction to protocol engineering. Available at http://cs.uccs.edu/~cs522/pe/pe.htm, 2006.

42 Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and reasoning on web services using process algebra. *IJBPIM*, 1(2):116–128, 2006. doi:10.1504/IJBPIM.2006.010025.

43 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, January 2019. doi:10.1145/3290343.

44 Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Process. Lett.*, 83(5):251–261, 2002. doi:10.1016/S0020-0190(01)00337-4.

45 Amrita Suresh and Nobuko Yoshida. The full version of this paper, 2025.

46 Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An Interaction-based Language and its Typing System. In *PARLE'94*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413, 1994. doi:10.1007/3-540-58184-7_118.

47 Martin Vassor and Nobuko Yoshida. Refinements for multiparty message-passing protocols: Specification-agnostic theory and implementation. In Jonathan Aldrich and Guido Salvaneschi, editors, *38th European Conference on Object-Oriented Programming, ECOOP 2024, September 16-20, 2024, Vienna, Austria*, volume 313 of *LIPIcs*, pages 41:1–41:29. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ECOOP.2024.41.

48 Malte Viering, Raymond Hu, Patrick Eugster, and Lukasz Ziarek. A Multiparty Session Typing Discipline for Fault-Tolerant Event-Driven Distributed Programming. *Proc. ACM Program. Lang.*, 5(OOPSLA), October 2021. doi:10.1145/3485501.

49 Daniel M. Yellin and Robert E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997. doi:10.1145/244795.244801.

A Proofs from §3

▶ **Lemma 14.** *An execution (e, ν) is causally equivalent to an i-RSC execution iff the associated conflict graph* $\mathsf{cgraph}(e, \nu)$ *is acyclic.*

Proof. The left to right implication follows from two observations: first, two causally equivalent executions have isomorphic conflict graphs. Secondly, the conflict graph of an RSC execution is acyclic, because for an RSC execution and vertices χ_1, χ_2 in the conflict graph, $\chi_1 \to_{e,\nu} \chi_2$ if there is $j_1 \in \chi_1$ and $j_2 \in \chi_2$ such that $j_1 \prec_{e,\nu} j_2$. Moreover, if there is more than action in either χ_1 or χ_2, for i-RSC executions by definition, $min(\chi_1) \prec_{e,\nu} min(\chi_2)$. Therefore, if there is a cycle in the conflict graph, then this would imply $min(\chi_2) \prec_{e,\nu} min(\chi_1)$, which would be a contradiction.

For the converse direction, let us assume that a conflict graph associated to $e = a_1 a_2 \dots a_n$ is acyclic. Let us consider the associated communication set ν. Let $\chi_1 \ll \cdots \ll \chi_n$ be a topological order on ν. Let $e' = \chi_1 \cdots \chi_n$ be the corresponding RSC execution, and ν' the communication set associated to e' that is RSC.

Let σ be the permutation such that $e' = a_{\sigma(1)} \cdots a_{\sigma(n)}$. Following the proof idea in [17], we show that e is causally equivalent to e'. Let j, j' be two indices of e, and let us show that $j \prec j'$ iff $\sigma(j) \prec \sigma(j')$.

We have that $\{j, j'\}$ is a matching pair in e, iff, by construction, $\{\sigma(j), \sigma(j')\}$ is a matching pair in e'. If $\{j, j'\}$ is not a matching pair of e, then let χ and χ' be the interactions containing j and j' respectively. Since $j \prec j'$, there is an arrow between χ and χ' in the conflict graph, and moreover a_j and $a_{j'}$ cannot commute. Note that there is an arrow in the conflict graph of e' as well. Since the conflict graph is acyclic, we have $\sigma(j) \prec \sigma(j')$. ◀

▶ **Lemma 16.** *$\mathcal{S}$ is i-RSC if and only if for all $e \in \mathsf{executions}(\mathcal{S})$ and $\nu \in \mathsf{Comm}(e)$, (e, ν) is not a borderline violation.*

Proof. By definition, if there exists an execution (e, ν) in system $\mathcal{S}$ such that it is a borderline violation, then $\mathcal{S}$ is not i-RSC. Conversely, if $\mathcal{S}$ is not i-RSC, let (e, ν) be (one of) the shortest execution that is not causally equivalent to an i-RSC execution. Then, $e = e' \cdot a$ such that for all $\nu' \in \mathsf{Comm}(e')$, we have (e', ν') is equivalent to an i-RSC execution. Let ν'' be the communication set of e' such that it is a subset of the communication set ν. Let (e'', ν'') be the i-RSC execution that is causally equivalent to (e', ν''). Then, there exists an execution $\hat{e}$ such that $(\hat{e}, \nu)$ is an execution of $\mathcal{S}$. Moreover, if a is a send action, then $(\hat{e}, \nu)$ is i-RSC which is a contradiction. Therefore, (e, ν) is a borderline violation. ◀

▶ **Lemma 17.** *Let $\mathcal{S}$ with $\mathsf{product}(\mathcal{S}) = (Q, \Sigma, Ch, \mathsf{Act}, \delta, q_o)$. There is a non-deterministic finite state automaton $\mathcal{A}_{bv}$ computable in time $\mathcal{O}(|Ch|^3|\Sigma|^2)$ such that $\mathcal{L}(\mathcal{A}_{bv}) = \{e \in \mathsf{Act}_{nr}^*.\mathsf{Act}_? \mid \exists \nu \in \mathsf{Comm}(e)$ such that (e, ν) is a borderline violation$\}$.*

Proof. Let $\mathcal{A}_{bv} = (Q_{bv}, \delta_{bv}, q_{0,bv}, \{q_f\})$, with $Q_{bv} = \{q_{0,bv}, q_f\} \cup (Ch \times \mathsf{Act} \times \{0, 1\})$, and for all $a, a' \in \mathsf{Act}_{nr}$, for all $c \in Ch$, $m, m' \in \Sigma$:

1. $(q_{0,bv}, a, q_{0,bv})$: this is a loop on the initial state that reads all interactions until the chosen send message
2. $(q_{0,bv}, \mathsf{pq}!m, (\mathsf{pq}, \mathsf{pq}!m, 0))$: this is the transition where we non-deterministically select the send message that is matched to the final reception to be borderline.
3. In case we do not consider out-of-order errors, we add the following step: $((\mathsf{c}, a, 0), a', (\mathsf{c}, a, 0))$, if $\mathsf{ch}(a') \neq \mathsf{c}$: again loop for every communication but we do not accept any further communication on the channel c in order to stay borderline. Note that this step is skipped if we consider the general case with out-of-order errors as we can have matched pairs between a matched send and receive action.
4. $((\mathsf{c}, a, 0), a', (\mathsf{c}, a', 1))$, if $\mathsf{proc}(a) \cap \mathsf{proc}(a') \neq \emptyset$: here, the second interaction that will take part in the conflict graph cycle is guessed. We ensure there is a process in common with a for there to be an edge between them.
5. $((\mathsf{c}, a, 1), a', (\mathsf{c}, a, 1))$, if $\mathsf{ch}(a') \neq \mathsf{c}$: once again a loop for every interaction.
6. $((\mathsf{c}, a, 1), a', (\mathsf{c}, a', 1))$, if $\mathsf{proc}(a) \cap \mathsf{proc}(a') \neq \emptyset$: the next vertex (or vertices) (if any) of the conflict graph is guessed.
7. $((\mathsf{c}, a, 1), \mathsf{pq}?m', q_f)$, if $\mathsf{proc}(a) \cap \mathsf{proc}(\mathsf{pq}?m') \neq \emptyset$: finally, an execution is accepted if it closes the cycle.

Moreover, each transition of $\mathcal{A}_{bv}$ can be constructed in constant time, so $\mathcal{A}_{bv}$ can be constructed in time $\mathcal{O}(|Ch|^3|\Sigma|^2)$. ◀

▶ **Lemma 18.** *Let $\mathcal{S}$ be a FIFO system. There exists a non-deterministic finite state automaton $\mathcal{A}_{rsc}$ over $\mathsf{Act}_{nr} \cup \mathsf{Act}_?$ such that $\mathcal{L}(\mathcal{A}_{rsc}) = \{e \cdot \mathsf{pq}?m \in \mathsf{Act}_{nr}^*.\mathsf{Act}_? \mid e \cdot \mathsf{pq}?m \in \mathsf{executions}(\mathcal{S})$ and $\exists \nu \in \mathsf{Comm}(e)$ such that (e, ν) is an i-RSC execution$\}$, which can be constructed in time $\mathcal{O}(n^{|\mathbb{P}|+2}|Ch|^2 \times 2^{|Ch|})$, where n is the size of $\mathcal{S}$.*

Proof. Let $\mathcal{A}_{rsc} = (Q_{\text{rsc}}, \delta_{\text{rsc}}, q_{0,\text{rsc}}, \{q_f\})$ be the non-deterministic automata, with $Q_{\text{rsc}} = Q \times (\{\varepsilon\} \cup Ch) \times 2^{Ch} \cup \{q_f\}$. We define the transitions as follows:

1. First, while performing the action $a \in \mathsf{Act}_{nr}$, $(q, \chi, S) \xrightarrow{a} (q', \chi', S')$ if
   - $(q, v) \implies (q', v')$ in the underlying transition system, for some buffer values v, v' and for all $\mathsf{c} \in Ch$, $v_c \neq \emptyset$ iff $\mathsf{c} \in S$ and $v'_c \neq \emptyset$ iff $\mathsf{c} \in S'$, and
   - this condition is added in the absence of out-of-order errors: if a $= \mathsf{pq}!?m'$, then $\mathsf{pq} \notin S$, and
   - either $\chi = \chi'$, or $a = \mathsf{c}!m$ and $\chi' = \mathsf{c}$
2. Second, while performing the action $a = \mathsf{pq}?m$, we have $(q, \chi, S) \xrightarrow{a} q_f$ if $\chi = \mathsf{pq}$ and $(q, a, q') \in \delta_{\mathcal{S}}$ for some q'.

Each transition of $\mathcal{A}_{rsc}$ can be constructed in constant time. An upper bound on the number of transitions can be computed as follows: if $(q, \chi, S) \xrightarrow{a} (q', \chi', S')$ is a transition, then q and q' only differ on at most two machines (the one that executed the send, and the one that executed the receive), so there are at most n^2 different possibilities for q' once q and a are fixed. There are at most two possibilities for χ' once χ and a are fixed, and S' is fully determined by S and a. Finally, there are $n^{|\mathbb{P}|}(1 + |Ch|) \times 2^{|Ch|} \times 2 \times |Ch|$ possibilities for a choice of the pair $((q, \chi, S), a)$. ◀

▶ **Theorem 19.** *Given a system $\mathcal{S}$ of size n, deciding whether it is an i-RSC system can be done in time $\mathcal{O}(n^{|\mathbb{P}|+2}|Ch|^5 \times 2^{|Ch|} \times |\Sigma|^2)$.*

Proof. The set of borderline violations of a system $\mathcal{S}$ can be expressed as $\mathcal{L}(\mathcal{A}_{rsc}) \cdot \mathsf{Act}_{nr} \cap \mathcal{L}(\mathcal{A}_{bv})$. Therefore, checking for inclusion in i-RSC reduces to checking the emptiness of this intersection, which can be done in time $\mathcal{O}(n)$. ◀